SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Dinko Ždravac

ODREĐIVANJE SLIJEDNOSTI TEKSTA METODAMA DUBOKOG UČENJA

Diplomski rad

Voditelj rada:
dr. sc. Tomislav Šmuc

Zagreb, 2018

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Zadatak prepoznavanja slijednosti teksta (eng. *Recognizing Textual Entailment*) sastoji se u određivanju logičke povezanosti para tekstova: premise i hipoteze. Potrebno je odrediti povlači li istinitost premise istinitost hipoteze, jesu li dva teksta u kontradikciji ili im je odnos neutralan.

Prepoznavanje slijednosti teksta klasifikacijski je problem strojnog učenja. U ovom radu uspoređujemo dosadašnji pristup rješavanju tog problema, baziran na strojnom učenju s ručnim probirom značajki, sa suvremenim tehnikama baziranima na modelima dubokog učenja (neuronske mreže), ističući njihove prednosti. Također, objašnjava se važnost prepoznavanja slijednosti teksta u sklopu drugih problema obrade prirodnog jezika strojnim učenjem. Iznose se osnovni koncepti teorije strojnog i dubokog učenja, u opsegu u kojem je potrebno za razumijevanje terminologije i načina rješavanja problema.

Bit će opisana 3 postojeća modela dubokog učenja, predstavljena u nedavno objavljenim znanstvenim radovima, koji postižu najbolje performanse na tom zadatku, a metodološki pokrivaju više različitih pristupa prepoznavanja slijednosti. Modeli su pojašnjeni po logičkim cjelinama i uspoređeni međusobno, dajući detaljniji uvid u rješavanje zadatka prepoznavanja slijednosti suvremenim tehnikama. Među predstavljenim modelima je ESIM model ([11]) koji postiže najbolje performanse na zadatku prepoznavanja slijednosti u vrijeme pisanja ovog rada.

Posljednje poglavlje predstavlja praktičan aspekt rada. Kao polazna točka iskorištena je postojeća Keras implementacija ([55]) spomenutog ESIM modela. Polazna implementacija postizala je suboptimalne rezultate jer je u više komponenti strukturno nedosljedna ESIM modelu iz rada prema kojem je napravljena. Stoga je nad implementacijom napravljena dorada arhitekture i promjena hiperparametara modela tako da bude usklađena s kodom originalnog ESIM modela. Usklađenost dorađenog Keras ESIM-a provjerena je i potvrđena postizanjem rezultata bliskih onima originalnog rada. S tom dorađenom verzijom Keras ESIM-a rađeni su sljedeći eksperimenti.

Predstavljene su tehnike Bayesove optimizacije hiperparametara na tom modelu. Ovo ima za cilj poboljšati rezultate te predstavlja suvremenu tehniku optimizacije modela specifičnu dubokom učenju. Optimizirani model postiže poboljšanje od 0.5%, što stavlja model još bliže performansama referentnog modela. Naposljetku, testiran je prijenos znanja mo-

dela provjerom nad nedavno objavljenim skupovima podataka velikog volumena. Također, izvršeno je treniranje modela nad kombinacijom postojećih skupova, evaluirajući prijenos znanja između skupova te analizirajući performanse po kategorijama. Keras ESIM postiže gotovo jednako dobre performanse kao referentni ESIM model, zaostajući na većini kategorija samo 1-2%, a na nekima i prestižući referentni model.

Poglavlje 1

Prepoznavanje slijednosti teksta

U ovom poglavlju opisujemo što je problem prepoznavanja slijednosti teksta, koje su njegove primjene i specifičnosti u odnosu na ostale zadatke obrade prirodnog jezika. Iznijet ćemo na koji je način zadatak prepoznavanja slijednosti rješavan dosad klasičnim metodama (strojno učenje) te po čemu se razlikuje suvremeni pristup baziran na dubokom učenju i koje prednosti nudi. Naposljetku, navest ćemo skupove podataka nad kojima se vrši strojno učenje uz kratak opis pojedinog skupa i njegove karakteristike.

1.1 Opis problema

Važnost problema prepoznavanja slijednosti

Problem prepoznavanja slijednosti teksta (eng. *Recognising textual entailment - RTE*), za dani par tekstova, definiramo kao određivanje povlači li logička istinitost prvog teksta (*premise P*) logičku istinitost drugog teksta (*hipoteze H*) ([20]). Ukoliko je to istina, kažemo da su premisa i hipoteza u usmjerenom odnosu *slijednosti* (hipoteza *slijedi* iz premise). Nastavno na to, par tekstova može biti i u neutralnom odnosu te u kontradikciji.

Prepoznavanje slijednosti teksta klasifikacijski je problem strojnog učenja. Za dani par tekstova, zadatak algoritma strojnog učenja jest klasificirati dani par obzirom na njihov odnos slijednosti u jednu od klasa: slijednost, kontradikcija ili neutralan odnos.

Problem je bitan kao sastavni potproblem mnogih drugih zadataka obrade prirodnog jezika. Sustavi za sumarizaciju teksta, automatsko odgovaranje na pitanja, strojno prevođenje, ekstrakciju informacija te dohvaćanje informacija, kao dio postupka trebaju moći prepoznati slijednost ([20], [19]). Sustav za automatsko odgovaranje na pitanja može ponuditi odgovor na pitanje, prepoznajući koji od dostupnih tekstova slijedi iz pitanja.

Također, zadatak prepoznavanja slijednosti koristi se za provjeru rezultata uspješnosti drugih zadataka obrade prirodnog jezika. Model za sumarizaciju teksta može ukloniti su-

višnu rečenicu iz teksta ukoliko ona slijedi iz preostalih rečenica izvornog teksta ([19]). Prijevod dobiven strojnim prevođenjem mora biti ekvivalentan izvornom tekstu, što znači da oba teksta moraju slijediti jedan iz drugoga ([20]).

Višeizražajnost (ista situacija može se opisati različitim frazama) i višeznačnost (pojedini izraz može imati različita značenja, ovisno o kontekstu u kojem je promatran) prirodnog jezika predstavljaju izazov računalnim modelima jer oni često nemaju kontekst ili pozadinsko socijalno znanje potrebno za raspoznavanje finih nijansi ljudskog izražavanja ([20]). Učestalnost nesporazuma u ljudskoj komunikaciji ukazuje na kompleksnost tog zadatka, čak i za ljude. Blizak problem određivanju slijednosti jest onaj prepoznavanja ekvivalencije dvaju tekstova ([19]). Za par tekstova kažemo da su ekvivalentni ako prenose isto značenje; ono što u svakodnevnom izražavanju zovemo parafraziranje.

Definicija određivanja slijednosti

Premisa i hipoteza mogu se sastojati od više rečenica, no za potrebe rasprave, bilo koju od njih možemo nazvati rečenicom, čak i ako se sastoji od više njih. Postoji više definicija tekstualne slijednosti, ovisno o kontekstu. Preciznija definicija postoji u lingvističkoj literaturi ([13]), a govori da premisa povlači hipotezu ako je hipoteza istinita uvijek kada je premisa istinita (uključujući sve moguće scenarije).
Na primjer, uzmimo par rečenica (preveden primjer iz [10])

- Nogometna utakmica s nekoliko muških igrača.
- Muškarci igraju određeni sport.

Razmatramo li ovu definiciju, nema slijednosti između navedenih premise i hipoteze jer postoje scenariji u kojima bi premisa bila istinita, a hipoteza ne. Takav scenarij je lako moguć u slučaju odvijanja dvije utakmice različitih sportova istovremeno, obje s muškim natjecateljima.

Za potrebe ovog rada, razmatrat ćemo nešto slobodniju definiciju slijednosti, za koju je dovoljno da je hipoteza *istinita s visokom vjerojatnošću*, ako je promatrana u kontekstu definiranom premisom. Tako gledano, kontekst je zadan premisom (odvija se nogometna utakmica s muškim igračima). Ovaj par rečenica uzet je iz SNLI skupa ([10]) gdje ga je većina (5 od 5) ocjenjivača označila kao slijednost (eng. *entailment, E*).

Ovo je dovoljno dobro za većinu praktičnih potreba ([19], [26]) jer većina navedenih primjena, kao npr. provjera sumarizacije teksta, razmatraju samo kontekst zadan premisom za određivanje slijednosti. Za modele gdje je potrebno, može se primijeniti preciznija definicija.

Razmotrimo još par rečenica koje su u kontradikciji:

- Crni trkaći automobil kreće ispred gomile ljudi.

- Osoba vozi duž usamljene ceste.

Ako nema *slijednosti niti kontradikcije*, rečenice su u *neutralnom odnosu*.

- Stariji i mlađi muškarac su nasmiješeni.
- Dvojica nasmiješenih muškaraca se smiju mačkama koje se igraju na podu.

1.2 Tradicionalni pristup

Tradicionalnim pristupom smatramo prvenstveno pristupe koji se baziraju na modelima s ručnim probirom značajki, tzv. inženjeringu značajki (eng. *feature engineering*) za izgradnju klasifikatora. *Značajka* u kontekstu strojnog učenja je mjerljivo svojstvo ili karakteristika ulaznog podatka na temelju koje model može učiti. Na primjeru određivanja slijednosti, korisna značajka može biti broj zajedničkih riječi ili slogova dvaju rečenica.

Za uspješnu konstrukciju modela strojnog učenja, potrebno je najprije pripremiti podatke za obradu. Taj proces sastoji se od nekoliko koraka:

- Priprema skupa podataka: prolazak kroz skup podataka te provjera ispravnosti svih unosa (vrijednosti koje nedostaju, konzistentnost unosa kroz čitav skup podataka i sl.).
- Normalizacija numeričkih podataka kako bi bili pogodni za statističku obradu/ulaz modelu.
- Označavanje podataka odgovarajućim oznakama.
- Odabir značajki na temelju kojih će računalni model zaključivati.

Inženjering značajki obuhvaća sve procese koji se odnose na selekciju i procjenu relevantnosti pojedinih značajki. Ne znamo unaprijed koje karakteristike ulaznih podataka najviše utječu na uspješno predviđanje rezultata. Treniranjem modela u iteracijama te promjenom značajki koje dajemo modelu omogućuju nam da empirijski utvrdimo koje od njih utječu pozitivno, a koje negativno na krajnji uspjeh modela. Ovo zahtijeva ponavljanje procesa treniranja sve dok ne optimiziramo rezultat obzirom na podatke koje imamo. Budući se radi o probiru podskupa značajki podataka, to može biti dugotrajan proces, a na kraju smo ograničeni vremenom i resursima (nije moguće obraditi sve varijacije u slučaju velikog broja značajki).

Kompleksne probleme kao što je obrada prirodnog jezika ne može se svesti na nekoliko desetaka, nekada čak ni stotina značajki. To rezultira ograničenim modelima koji dobro nauče reprezentaciju podataka u skupovima za treniranje, ali imaju slabu generalizaciju na dosad neviđene podatke, posebice one koji se znatno razlikuju u odnosu na naučene

primjere. Također, za primjenu nad drugim tipom podataka (npr. drugi jezik) potrebno je ponovno vršiti odabir značajki jer se model oslanja na određenu unutarnju strukturu podataka nad kojima je konstruiran.

Primjer ručne selekcije značajki imamo kod leksičkog klasifikatora za prepoznavanje slijednosti tekstova u [10]. Navedeni klasifikator zaključuje na temelju 6 značajki, od kojih su 3 leksičke (razlika duljina premise i hipoteze, postotak preklapanja riječi te ocjena preklapanja n-grama), a 3 neleksičke (oznake za *uni-* i *bi-grame* u rečenicama te dijeljeni *uni-* i *bi-grami* za iste uloge u rečenicama). Česte oznake su tzv. POS oznake (eng. *part-of-speech*) koje označavaju koja riječ ima pojedinu ulogu u rečenici (subjekt, predikat, objekt, atribut itd.). Navedene značajke obično je moguće automatski generirati ([45]), ali neke je potrebno formirati (ili barem provjeriti) ljudski. Stoga, ni svaka dva skupa podataka nemaju primjenjive iste značajke, pa time ni modele.

U odnosu na to, modeli dubokog učenja ne zahtijevaju ručni probir značajki jer sami uče hijerarhijske reprezentacije podataka. Time se preskače pedantan i greškama podložan proces ručnog probira značajki od strane čovjeka.

1.3 Suvremeni pristup baziran na dubokom učenju

Duboko učenje uči hijerarhijske reprezentacije podataka, formirajući kompleksnija značajke na temelju kompozicije hijerarhijski jednostavnijih značajki. Kompleksni problemi razlažu se na više slojeva apstrakcije koje model uči postupno. Model sâm konstruira značajke nad kojima vrši učenje umjesto korištenja unaprijed probranih značajki (npr. slogovi riječi, POS oznake) te odabrane značajke proslijeđuje sljedećim slojevima. Tako se kroz slojeve modela dubokog učenja uče međureprezentacije podataka, postupno učeći sve kompleksnije koncepte. Na primjeru prepoznavanja slika, model najprije uči prepoznavati rubove objekata, potom sljedeći sloj gradi nad tim apstrakcijama te uči kuteve i siluete, sljedeći uči različite oblike pa sve do prepoznavanja čitavih objekata i njihove klasifikacije. Jedan sloj smatramo funkcijskim preslikavanjem, a slaganje slojeva kompozicijom funkcija (ulančane transformacije podataka).

Učenje značajki podataka pomoću različitih slojeva apstrakcije omogućuje modelima da nauče kompleksna funkcijska preslikavanja bez potrebe za ručnom selekcijom značajki ([4], [9], [35]). Taj pristup zovemo *učenje s kraja na kraj* (eng. *end-to-end learning*) jer model sâm uči međureprezentacije podataka, bez potrebe za ljudskom intervencijom u međukoracima. Ono predstavlja konceptualnu razliku u odnosu na tradicionalni pristup jer nije potrebno birati značajke svakog skupa podataka pojedinačno, budući ih model može sam naučiti sloj po sloj. Izgradnja modela svodi se na konstrukciju pojedinih slojeva koji rade ono što se od njih intuitivno očekuje - model za neuralnu pažnju zaista skreće pažnju sljedećeg sloja na relevantne fraze pojedine rečenice.

Ovo čini duboko učenje primjenjivim na razne domene: model za zaključivanje o slijednosti može podjednako dobro učiti zaključivanje o različitim jezicima, ovisno na kojem jeziku mu proslijedimo ulaz, bez potrebe da za drugi jezik modificiramo model birajući značajke ipočetka, što predstavlja značajan odmak od tradicionalnog pristupa koji se oslanja na inženjering značajki i brojne međukorake koji ovise o specifičnostima ulaza.

Obzirom da uče sami probirući značajke, za uspjeh modela dubokog učenja ključna je velika količina dobro označenih podataka koji će omogućiti da model ima dovoljno informacija da probere relevantne koncepte iz šume podataka. Stoga duboko učenje pokazuje dobru primjenu nad kompleksnim zadacima nadziranog učenja s velikim brojem parametara ([28]). Pored toga, modeli dubokog učenja dobro skaliraju s dodavanjem slojeva (dubina modela) te povećanjem skupa za učenje, dok je kod tradicionalnih modela uočljiva stagnacija bez obzira na količinu podataka obzirom na izabrane značajke.

1.4 Skupovi podataka

SNLI

2015. objavljen je Stanford Natural Language Inference Corpus (SNLI, [10]), najveći skup podataka prilagođen za učenje prepoznavanja slijednosti teksta dotad. Ovaj skup je dva reda veličine veći od dotad najvećeg skupa u kojem su sve rečenice napisane od strane čovjeka ([36]). SNLI se sastoji od 570K parova rečenica (premise i hipoteze), s oznakama iz skupa {slijednost, neutralno, kontradikcija} (eng. *entailment, neutral, contradiction*) koje govore o odnosu slijednosti između premise i hipoteze za taj par. SNLI ima i kvalitativnu prednost jer su svi parovi rečenica u skupu konstruirani od strane čovjeka, nasuprot skupovima poput [17] gdje su parovi premise i hipoteze automatski izvučeni iz novinskih članaka (naslov kao premisa, prva rečenica teksta kao hipoteza koja slijedi) ili [54], gdje su parovi dobiveni algoritamskim skraćivanjem ljudski pisanih opisa slika.

Premise u SNLI skupu su preuzete iz postojećeg skupa Flickr30k ([54]). To su rečenice napisane od strane ljudi, kojima je dan zadatak da opišu danu fotografiju preuzetu sa Flickr-a[1]. Međutim, ovdje hipoteza nije dobivena automatskim skraćivanjem rečenice kao u [54]. Za konstrukciju hipoteze, ljudima je dan zadatak da za zadanu premisu napišu tri rečenice: jednu koja je *sigurno istinita* kada je prva rečenica istinita, drugu koja *može* biti istinita te treću koja *sigurno nije istinita* ako je zadana rečenica istinita. Na taj način dobiveni su parovi sa sva tri odnosa slijednosti. Time je SNLI najveći skup u kojem su premisa i hipoteza u potpunosti napisane od strane čovjeka, sa zadanim odnosa slijednosti na umu prilikom pisanja. 10% ukupnog broja rečenica provjereno je tako što su još 4 osobe, tzv. *ocjenjivača*, dale svoj sud o slijednosti para rečenica. Za 98% ocijenjenih rečenica ba-

[1] `https://www.flickr.com/`

rem troje ocjenjivača slaže se oko kategorije slijednosti dok 58% parova ima jednoglasnu odluku ocjenjivača.

SNLI skup postao je standard mjerenja uspješnosti modela prepoznavanja slijednosti teksta, pa ga time smatramo i u ovom radu ([41], [39], [11]). Svi modeli predstavljeni u ovom radu koriste SNLI za treniranje i testiranje uspješnosti. SNLI skup distribuiran je s unaprijed napravljenom podjelom na trening, validacijski i testni skup. Ta se podjela koristi u svim eksperimentima jer su se autori skupa pobrinuli da trening, validacijski i testni skup sadrže reprezentativne primjere (podjednak broj rečenica za svaku klasu slijednosti).

MNLI

MNLI (eng. *Multi-Genre Natural Language Inference* [52]) skup podataka izgrađen je u svrhu učenja i unaprijeđenja modela strojnog učenja na zadatku prepoznavanja slijednosti teksta po uzoru na SNLI skup. Kako je SNLI skup konstruiran rečenicama iz većinom istog rečeničnog konteksta (opisi slika), nedostaju mu šire jezične karakteristike: različita vremena zbivanja (jučer, sutra), izražavanje znanja, vjerovanja i modaliteta (hipotetske situacije). Zbog toga je SNLI ograničen kao generalni skup za treniranje i provjeru modela prirodnog jezika.

MNLI se sastoji od 433K parova rečenica podijeljenih u 10 kategorija govornog i pisanog engleskog jezika. Kategorije su zamišljene da predstave punu raznolikost načina na koji koristimo jezik u današnje vrijeme.

9 od 10 kategorija tekstova preuzeto je iz *Open American National Corpusa*[2], velikog skupa tekstova raznih žanrova i transkripata govornog jezika u elektroničkom obliku. To su razgovori licem u lice (*"Face-to-face"*), izvještaji, govori i objave za medije preuzete s javnih vladinih stranica (*"Government"*), pisma filantropske ogranizacije (*"Letters"*), javni izvještaj o terorističkim napadima koji su se dogodili u SAD-u 11. rujna 2001. (*"9/11"*), nekoliko djela na temu dječjeg razvoja i tekstilne industrije (*"OUP"*), članci iz popularne kulture časopisa Slate (*"Slate"*), transkripti telefonskih razgovora (*"Telephone"*), turistički vodiči (*"Travel"*) i kratki članci o lingvistici (*"Verbatim"*). 10. kategorija, fikcija (*"Fiction"*) sastavljena je od tekstova iz nekoliko slobodno dostupnih radova suvremene fikcije ([52]).

Premise su uzete iz navedenih tekstova uz minimalne modifikacije - eliminaciju matematičkih formula, bibliografskih referenci i sl. Hipoteze su prikupljene na način sličan kao u SNLI: osobi je dana premisa uz uputu da sastavi tri rečenice, jednu koja je istinita kada je premisa istinita (slijednost), jednu koja je nužno neistinita u kontekstu premise (kontradikcija) i jedna za koju ne vrijedi nijedan od navedenih uvjeta (neutralan odnos). Svaki par rečenica u testnom i validacijskom skupu provjerile su dodatne 4 osobe tako što su dale

[2] `http://www.anc.org/`

svoje mišljenje o slijednosti parova. Rečenice za koje ne postoji konsenzus su distribuirane uz skup, ali ne koriste se u eksperimentima.

Moguće je testirati prenošenje znanja modela treniranog nad pojedinim kategorijama na druge, neviđene kategorije. Iz tog razloga, trening skup MNLI-a sadrži samo 5 kategorija, dok postoje dvije varijante testnog i validacijskog skupa. Testni i validacijski skup *usklađenih kategorija* sadrže 5 kategorija rečenica koje se nalaze u trening skupu. Nasuprot tome imamo *neusklađene* varijante testnog i validacijskog skupa s rečenicama iz preostalih 5 kategorija koje se ne nalaze u trening skupu.

Trening skup sadrži 392K parova, dok svaka kategorija testnog i validacijskog skupa sadrži po 2K parova. To daje po 10K primjera u testnom i validacijskom skupu usklađenih kategorija i još po 10K parova za testni i validacijski skup neusklađenih kategorija. Nijedna varijanta testnog skupa primjera (usklađena ni neusklađena) nisu javno objavljene jer su rezervirane za provjeru modela na Kaggle[3] natjecanjima.

[3] `https://www.kaggle.com/`

Poglavlje 2

Duboko učenje

Nakon upoznavanja s problemom prepoznavanja slijednosti teksta, predstavljamo metodologiju njegova rješavanja tehnikama strojnog učenja. Definirat ćemo što je učenje u kontekstu strojnog učenja te iznijeti pojmove relevantne za razumijevanje iznesenog problema i rješavanja RTE zadatka pomoću dubokog učenja. Potom ćemo uvesti pojam dubokog učenja, istaknuti po čemu se ono razlikuje u odnosu na klasično strojno učenje. Navest ćemo osnovnu arhitekturu neuronskih mreža za duboko učenje te uobičajene tehnike optimizacije i regularizacije koje služe pri konstrukciji modela dubokog učenja.

2.1 Strojno učenje

Osnovni pojmovi i način učenja

Strojno učenje definiramo na sljedeći način ([38]):

Definicija 2.1.1. *Računalni program uči na temelju iskustva E obzirom na klasu zadataka T i mjeru uspješnosti P ako njegova uspješnost nad zadatkom T raste s dostupnim iskustvom E (skup podataka) obzirom na mjeru uspješnosti P.*

Za primjer uzmimo zadatak igranja šaha: kažemo da računalo uči igrati šah (zadatak T) ako njegov postotak pobjeda protiv drugih igrača (mjera uspješnosti P) raste s količinom obrađenih informacija o postojećim mečevima (iskustvo E).

Da bi program učio, mora imati dostupan skup podataka (iskustvo E). Taj skup se sastoji od *primjera* odnosno *ulaza*, od kojih svaki ima određene značajke. Primjer obično označavamo n-torkom $\mathbf{X} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, čija je svaka komponenta x_i jedna značajka. Primjeri su u računalu reprezentirani tenzorima koji predstavljaju ulaz (slike, tekst i sl.). Program koji vrši učenje zovemo *model*. Model ima *ulaze* (podatke) i *izlaze* (brojeve, funkcije, složene tipove podataka).

Algoritme strojnog učenja možemo grubo podijeliti obzirom na iskustvo E prilikom učenja, na algoritme *nadziranog* i *nenadziranog* učenja. *Nadzirano učenje* naziv je za učenje kod kojeg su modelu dostupni označeni primjeri x s ciljnim vrijednostima y za svaki primjer, a učenje ima za cilj predvidjeti ciljnu vrijednost y za dani primjer x. Učenje se zove nadzirano jer ispravne vrijednosti imaju ulogu učitelja, dajući modelu ispravne informacije na temelju kojih uči. Primjer nadziranog učenja je prepoznavanje objekata na slikama, gdje je primjer za učenje x slika, a y su ispravne oznake na toj slici za objekte koje model treba prepoznati. Problem RTE je problem nadziranog učenja jer za svaki par premise i hipoteze, modelu strojnog učenja dajemo ispravnu vrijednost odnosa tih dvaju rečenica, na temelju kojih se vrši učenje.

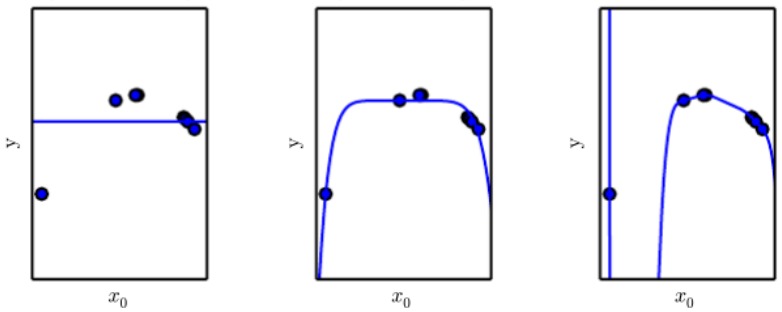
Kod *nenadziranog učenja* nemamo zabilježene ispravne klase za svaki ulaz, nego se ono oslanja na automatsko otkrivanje uzoraka u podacima, stoga se koristi za zadatke kao što su segmentacija podataka, otkrivanje izuzetaka i slične primjene. Postoje tipovi učenja koji kombiniraju oba pristupa (polu-nadzirano učenje, učenje nagrađivanjem (eng. *reinforcement learning*) i sl. ([28])) pa podjela na nadzirano i nenadzirano služi samo za grublju kategorizaciju. Različiti tipovi učenja obzirom na iskustvo E konceptualno se razlikuju te mnoge stvari nisu primjenjive iz jednog područja na druga. Pokrivanje svih koncepata izvan je dosega ovog rada. Kako je prepoznavanje slijednosti zadatak nadziranog učenja, u ovom radu uvodimo samo terminologiju i prakse oblikovanja modela strojnog učenja primjenjive na zadatke nadziranog učenja.

Obzirom na vrstu zadatka T koji rješavaju, algoritme nadziranog učenja dijelimo na *regresijske* i *klasifikacijske* modele. Regresijski model ima za cilj predvidjeti numeričku vrijednost na temelju postojećih vrijednosti. Primjer takvog algoritma je algoritam za predviđanje sljedeće vrijednosti u nizu brojeva, predviđanje kretanja cijene nekretnina i sl. Klasifikacijski model ima za cilj preslikati (*klasificirati*) ulaz x u odgovarajuću kategoriju (klasu) y. Model koji vrši klasifikaciju zovemo i *klasifikator*. Problem prepoznavanja slijednosti je upravo klasifikacijski problem, gdje je cilj razvrstati par rečenica u jednu od klasa koja opisuje njihov odnos slijednosti. Svi modeli koje obrađujemo u ovom radu su klasifikacijski modeli za RTE problem.

Želimo oblikovati model koji preslikava ulazne primjere x u njima ispravne vrijednosti y. Pretpostavljamo da postoji funkcija f koja dobro opisuje odnos između x i y. No, ne znamo oblik te funkcije. Znamo samo vrijednosti na podskupu domene, dostupnom skupu primjera za učenje. Drugim riječima, želimo aproksimirati funkciju f konstruirajući model kao funkciju f^*, što točnije iz dostupnog skupa primjera, koji zovemo *trening skup*. Tada za novi primjer x_k određujemo $z_k = f^*(x_k)$ pomoću te funkcije. Cilj je da z_k bude što bliži (ili jednak) stvarnom y_k, koji možemo, ali ne moramo znati.

Kao jednostavan primjer uzmimo linearnu regresiju, koja linearnim preslikavanjem aproksimira sljedeću vrijednost na temelju postojećih vrijednosti (gdje je x ulazni primjer):

$$f^*(x) = Wx + B \tag{2.1}$$



Slika 2.1: **Lijevo:** linearna funkcija. **Sredina:** polinom stupnja 4. **Desno:** polinom stupnja 9. Kako raste kapacitet modela, on bolje aproksimira vrijednosti na trening skupu. Međutim, na najdesnijem primjeru vidimo da se prekompleksan model pretrenira na postojeći skup primjera. Takav model vjerojatno neće dobro generalizirati na nove primjere jer i jednostavniji model, polinom stupnja 4, dobro aproksimira ciljnu funkciju. Izvor: [28]

f^* je funkcija koja ovisi o parametrima: *težinama W* (eng. *weights*) i *pomacima* odnosno *sklonostima B* (eng. *bias*), pa ju označavamo i kao $f^*(x|\theta)$, gdje s θ označavamo sve parametre modela (u ovom slučaju težine i sklonosti). Premda se funkcije različitih modela mogu znatno razlikovati, svaka funkcija oblikovana je parametrima koji određuju *familiju funkcija* koju model može opisati, kao što model linearne regresije može opisati familiju linearnih funkcija, ovisno o izboru parametara W i B.

Mogućnost modela da opisuje širok spektar funkcija zovemo *kapacitet modela*. Kompleksne probleme nije moguće dobro modelirati jednostavnim modelima. Za takve probleme potrebno je povećati kapacitet modela. To je moguće promjenom njegovog *prostora hipoteza*, tj. skupa funkcija koje može opisati. Slika 2.1 ilustrira to na primjeru regresije. Za aproksimaciju distribucije točaka u ravnini kao na slici, nije dovoljno koristiti linearnu regresiju. Dodavanjem dodatnih, polinomijalnih članova, funkcija f^* postaje polinom višeg reda te bolje aproksimira ciljnu funkciju distribucije primjera (polinomijalna regresija). Time smo promijenili prostor hipoteza ovog modela s prostora svih linearnih funkcija na prostor svih polinoma određenog reda i time povećali njegov kapacitet.

Sposobnost modela da dobro izvršava zadatak nad neviđenim podacima zovemo *generalizacija*. Model koji dobro nauči podatke nad kojima je treniran, a loše predviđa nad novim, neviđenim primjerima, smatramo lošim jer *loše generalizira*. Model koji nije dovoljno kompleksan, ne može dobro opisati ciljnu funkciju nad trening skupom jer je podtreniran (eng. *underfitting*). Ako je model prekompleksan, on može "naučiti" sve pravilnosti u trening skupu, uključujući *šum*; tada model postaje *pretreniran* (eng. *overfitting*), čime se smanjuje njegova sposobnost generalizacije (slika 2.1). Tada model postiže dobre rezultate nad trening skupom, ali loše rezultate na novim, neviđenim primjerima. Model postiže optimalne rezultate kada njegov kapacitet odgovara kompleksnosti problema koji opisuje.

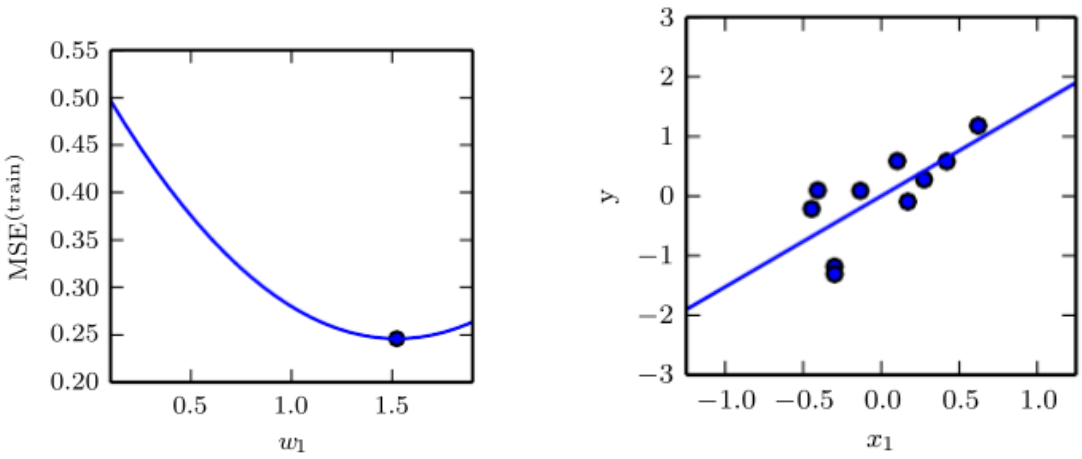
Tijekom učenja, kako bismo znali koliko je uspješan model, računamo *grešku* kao negativnu mjeru uspjeha modela. Greška *ovisi* o tome koliko je dobar model pa mjeru greške zovemo *funkcijom greške*. Budući da uspjeh modela ovisi o parametrima, funkcija greške je *funkcija parametara modela*. Dakle, kako bi greška bila minimizirana, potrebno je ažurirati parametre modela na odgovarajuće vrijednosti koje, za što više ulaza, griješe što manje. Greška se računa na skupu za učenje kao razliku između *stvarnog* i *točnog* izlaza modela za svaki ulaz. Oblik funkcije greške specifičan je zadatku. Kako bismo dobili metriku koliko je model uspješan nad cijelim skupom podataka (npr. za treniranje), koristimo općenitiju *funkciju cijene/troška* nad cijelim skupom primjera, koju označavamo s L. Dakako, računanje greške kao razlike stvarnog i postignutog ulaza moguće je samo kod nadziranog učenja, gdje postoje ispravne vrijednosti za primjere. Nenadzirano učenje koristi drugačije tehnike za mjerenje uspjeha modela.

Postoje mnoge varijante funkcije troška L, ovisno o tipu i zadatku strojnog učenja. Funkcija troška primjenjiva na danom primjeru linearne regresije je funkcija troška na temelju srednje kvadratne greške (eng. *mean squared error - MSE*), čija je formula na trening skupu $TRAIN$ dana jednadžbom (2.2). Očito, ova greška konvergira prema 0 kada je $z_i = y_i$ za velik broj primjera.

$$L_{MSE}^{(TRAIN)} = \frac{1}{|TRAIN|} \sum_i (z_i^{(TRAIN)} - y_i^{(TRAIN)})_i^2 \tag{2.2}$$

Promjena kapaciteta modela kako bi bolje pasao zadatku može se postići na više načina, od kojih je samo jedan povećanje kompleksnosti funkcije f^*. Proces učenja je optimizacija parametara θ kako bi se s povećanjem iskustva (promatranjem primjera) našla odgovarajuća konfiguracija parametara koja minimizira funkciju troška L. To se postiže optimizacijskim algoritmima koji koriste neku od tehnika pronalaska minimuma funkcije troška. Jedan od takvih je optimizacija na temelju smjera gradijenta, o čemu će više biti govora u poglavlju 2.3. Na slici 2.2a vidimo primjer minimizacije funkcije troška L, čiji su parametri (u ovom slučaju samo težina w_1) minimizirani, što rezultira dobrom aproksimacijom distribucije primjera linearnom regresijom f^* (najboljom koja se može postići *linearnom* familijom funkcija, slika 2.2b).

Ocjena generalizacije modela provjerava se uspjehom nad primjerima koje model nije vidio tijekom treninga. Jedan od načina je izdvajanjem primjera prije treniranja nad kojima ćemo samo testirati. To je moguće kada je skup dostupnih primjera dovoljno velik da ne gubimo mnogo izdvajanjem podskupa primjera samo za testiranje. Budući da model uči nad trening skupom, s vremenom odabire vrijednosti parametara tako da model ima sklonost prema primjerima iz trening skupa, što mu može smanjiti mogućnost generalizacije na neviđene primjere. Zato za računanje točnosti koristimo odvojen skup nad kojim model nije treniran. No, kako model ažuriramo s povratnim informacijama iz tog skupa (računajući performanse i ažurirajući komponente modela), imamo sklonost izabrati model koji je posebno dobro prilagođen i testnom skupu, pored trening skupa. Kako bi se



(a) Minimizacijom funkcije troška L obzirom na parametre postiže se manja greška modela.

(b) Manja greška modela znači da ciljna funkcija f^* bolje aproksimira distribuciju primjera (jer manje griješi).

Slika 2.2: Minimalna vrijednost težine w_1 rezultira dobrom aproksimacijom ciljne funkcije. Izvor: [28])

to izbjeglo, iz početnog skupa primjera se, prije učenja, odvaja dio primjera u *validacijski skup*, nad kojim provjeravamo performanse modela za vrijeme njegova oblikovanja, a dio primjera u *testni skup* nad kojim se provjerava uspjeh modela na kraju, kada je postignuta zadovoljavajuća točnost nad validacijskim skupom.

Ovaj pristup testiranju i treniranju zovemo podjela na trening, test i validacijski skup. Ukoliko bi razdvajanje skupa dostupnih primjera na navedene podskupove previše umanjilo trening skup, koristi se neka od drugih tehnika evaluacije (npr. k-struka unakrsna validacija, [28]).

Računanje greške je jedna metrika uspješnosti, ali obično se koriste još neke koje intuitivnije ilustriraju performanse modela za razne zadatke. Za klasifikacijske probleme, pa tako i RTE problem, jedna takva jest *točnost* (eng. *accuracy*), koja daje broj točno klasificiranih primjera od ukupnog broja primjera (jednadžba (2.3)). Ovo nije jedina niti najbolja mjera, ali se najčešće koristi za mjerenje performansi modela koji prepoznaju slijednost teksta ([11], [39], [41], [52], [10]). Odabir mjere uspješnosti ovisi o cilju i specifičnostima modela.

$$\text{točnost} = \frac{\#(\text{točno razvrstanih primjera})}{\#(\text{ukupno primjera})} \tag{2.3}$$

Navedena mjera uspjeha (točnost) primjenjiva je samo na klasifikacijske i slične probleme (klasifikacija s nedostajućim vrijednostima, transkripcija [28]) gdje je primjer točno

ili pogrešno svrstan (nema nijansi između toga), dok se za ostale vrste problema koriste nešto drugačije, nad njima primjenjive mjere uspjeha. Također, opisani model odgovara nadziranom klasifikacijskom modelu, a ne općenito, jer za strojno učenje postoji velik broj različitih modela nad kojima nisu primjenjive iste funkcije greške.

Pri konstrukciji modela, potrebno je paziti na odnos kompleksnosti modela i kompleksnosti zadatka. Želimo što jednostavniji model koji postiže optimalnu točnost za dani zadatak. Tehnike kojima se regulira kompleksnost modela zovemo *regularizacija*. To se, između ostalog, postiže uvođenjem dodatne funkcije koja ovisi o parametrima, a koja kažnjava kompleksnost modela. Često se u funkciju troška pored standardnog izračuna greške L dodaje i *regularizacijska funkcija* R koja ovisi o parametrima modela (jednadžba (2.4)).

$$\Phi(x, y, \theta) = L(x, y, \theta) + R(\theta) \tag{2.4}$$

Vidljivo je da, minimizacijom funkcije greške L, dobivamo manju vrijednost funkcije troška Φ. No, sada nam je bitno i da pazimo na regularizacijsku funkciju. Najčešće se primjenjuju regularizacijske funkcije R koje linearno ili kvadratno ovise o parametrima. Takve tehnike regularizacije zovemo $L1$ odnosno $L2$ regularizacija za linearni odnosno kvadratni član. Izbor regularizacijske funkcije, kao i mnoge druge stvari, specifičan je problemu koji rješavamo.

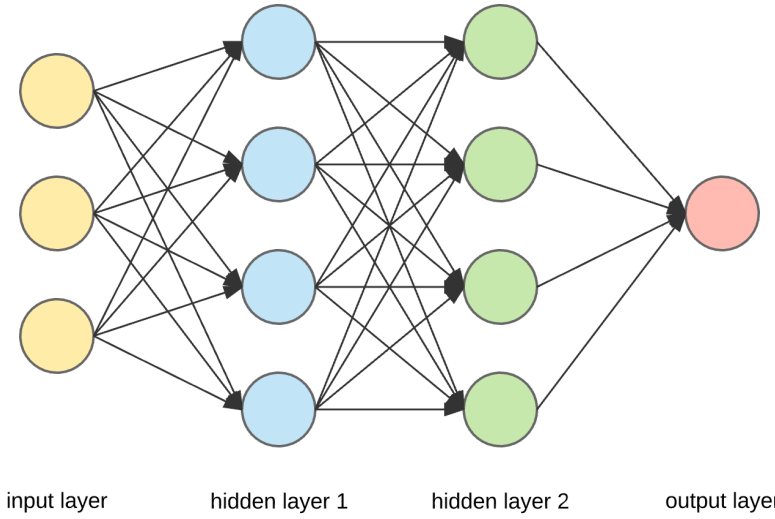
Ovakve meta-parametre kojima oblikujemo sam proces učenja te koji se *ne mijenjaju tijekom učenja* jer se ne mogu naučiti iz podataka od strane modela zovemo *hiperparametrima*. Hiperparametrima određujemo karakteristike modela koji uči, kao što su njegova kompleksnost ili brzina učenja. Spomenuta regularizacijska funkcija, funkcija troška kojom računamo grešku modela ili optimizacijska funkcija kojom nalazimo minimum funkcije troška primjeri su hiperparametara.

2.2 Neuronske mreže

Duboke neuronske mreže, poznate i kao unaprijedne neuronske mreže (eng. *feedforward neural network*) ili višeslojni perceptroni (eng. *multi-layer perceptron - MLP*) naziv je za arhitekturu modela strojnog učenja koja je sastavljena od više međusobno povezanih jedinica, tzv. neurona, posloženih u slojeve. Kompozicijom slojeva postiže se visok kapacitet modela za mnoge kompleksne zadatke te su neuronske mreže danas na mnogim područjima prestigle tradicionalno oblikovane modele strojnog učenja u uspješnosti ([41], [50], [3]). Ovdje dajemo kratki pregled arhitekture mreža, građe osnovnih jedinica i osnovne tehnike treniranja neuronskih mreža.

Arhitektura neuronskih mreža

Neuronska mreža zove se tako jer je sastavljena od velikog broja jednostavnih jedinica poslo-ženih u slojeve. Svaki sloj mreže sastoji se od pojedinačnih računskih jedinica, *neurona*. Neuron je funkcija koja preslikava vektor u skalar (slika 2.4). Komponente vektora koje predstavljaju ulaz neuronu su izlazi neurona prethodnih slojeva, a skalarni izlaz trenutnog neurona, u kombinaciji s ostalim neuronima istog sloja, predstavlja ulaze neuronima sljedećeg sloja, time čineći strukturu nalik mreži (slika 2.3).



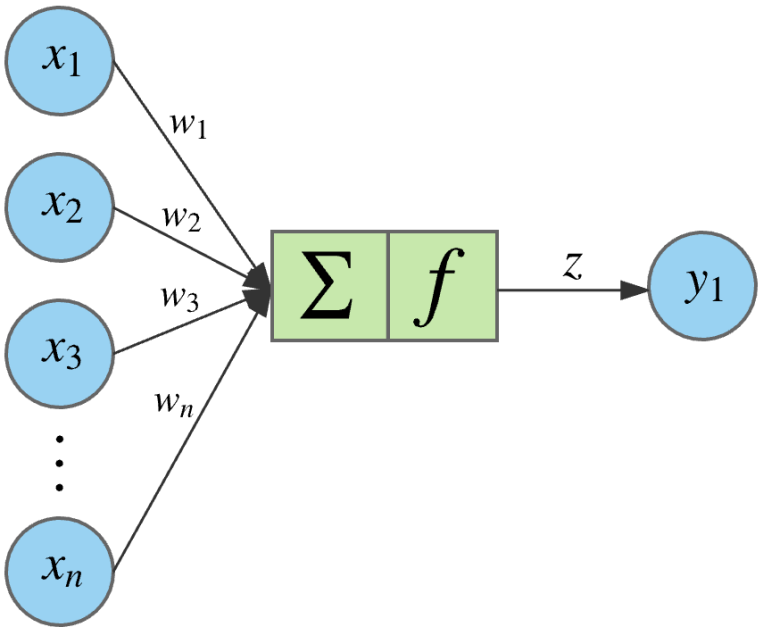
Slika 2.3: Građa neuronske mreže - neuroni povezani u više slojeva proslijeđuju informacije prema naprijed. Ovaj tip mreže zove se unaprijedna neuronska mreža (eng. *feed-forward neural network*). Izvor: [23]

Samim time, sloj je također funkcija, koja preslikava vrijednosti prethodnog sloja, proslijeđujući ih sljedećem sloju mreže. Mreža kao model je kompozicija ulančanih slojeva: ako s $f^{(1)}, f^{(2)}, \ldots, f^{(k)}$ označimo ulančane slojeve neuronske mreže, tada je neuronska mreža kao preslikavanje definirana kao

$$f(x) = f^{(k)}(f^{(k-1)}(\ldots f^{(1)}(x)\ldots)) \tag{2.5}$$

gdje je x neka reprezentacija ulaza (matrica).

Broj slojeva mreže zovemo *dubina* mreže. Korištenjem dubljih modela, s više ulančanih slojeva, mogu se modelirati vrlo kompleksne funkcije, odakle dolazi naziv *duboko učenje*. Prvi sloj mreže koji obrađuje ulazne podatke zovemo *ulazni sloj*, a posljednji sloj *izlaznim slojem*. Slojeve između njih zovemo *skrivenim slojevima* jer nemamo direktan pristup njihovim izlaznim vrijednostima (skrivene su od vanjskog promatrača).



Slika 2.4: Građa perceptrona s ulazima x_i, težinama w_i i pomacima b_i (nisu prikazani), aktivacijskom funkcijom f i izlazom y_1. Izvor: [23]

Transformacija podataka u sloju događa se na razini neurona. U njemu se linearno preslikava niz vrijednosti $x_1, x_2, \ldots, x_n$ sa n ulaza, u skalar z koji neuron šalje neuronu u sljedećem sloju (jednadžba (2.6), slika 2.4).

$$z_j = \sum_i w_i x_i + b_j \tag{2.6}$$

Ažuriranjem težina w_i i sklonosti b_i mijenja se funkcijsko preslikavanje neurona pa time i modela. Treniranje modela jest ažuriranje tih parametara kako bi se što bolje aproksimirala ciljna funkcija f. Međutim, jednadžba (2.6) je linearno preslikavanje, a komponiranjem linearnih preslikavanja možemo modelirati samo linearne funkcije. Kako bi kapacitet mreže bio veći, uvodi se nelinearno preslikavanje na izlaznu vrijednost svakog neurona: *aktivacijska funkcija*.

Aktivacijska funkcija primjenjuje se na izlaz neurona, dajući sloj nelinearnosti koji povećava kapacitet modela. Dokazano je da unaprijedna neuronska mreža s jednim slojem s dovoljnim brojem neurona te određenim ograničenjem na aktivacijsku funkciju može aproksimirati bilo koju neprekidnu funkciju realne varijable na zatvorenom i ograničenom podskupu skupa $\mathbb{R}^n$ ([18], [30]).

Aktivacijska funkcija ima još jednu posljedicu na mrežu. Osim dodavanja nelinearnosti u model, aktivacijska funkcija svojim oblikom određuje koji neuroni će proslijediti

informacije i u kojoj mjeri. Najprimitivniji oblik je linearna funkcija $f(x) = x$, koja transformira ulaz proporcionalno dobivenoj vrijednosti u jedinici (neuronu). Sofisticiraniji i češće korišten oblik je *rektificirana linearna funkcija* ([27], [33]), koja vrijednosti ispod granice preslikava u 0, a iznad, linearno (jednadžba (2.7), slika 2.5a).

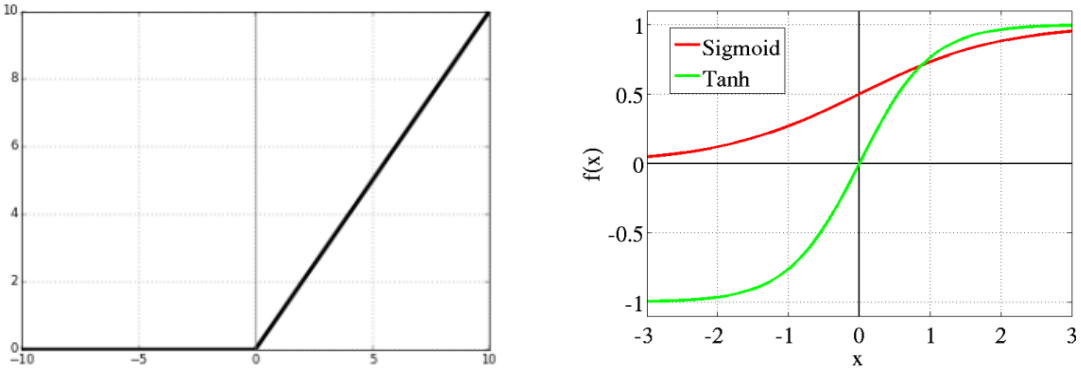
Neurone koji koriste odgovarajuću aktivacijsku funkciju obično zovemo po toj funkciji, pa tako mreža (ili sloj) koja implementira neurone s rektificiranom linearnom funkcijom zovemo mrežom (ili slojem) s *ReLU (eng. ReLU = Rectified Linear Unit)* aktivacijom. Neuroni ispod granične vrijednosti ReLU funkcije neće proslijediti svoju vrijednost u sljedeći sloj (proslijedit će 0), kao da nisu aktivni, odakle naziv aktivacijska funkcija.

$$RELU(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{2.7}$$

Ovisno o primjeni, moguće su i varijante logističke sigmoidne (jednadžba (2.8), slika 2.5b) ili tangens hiperbolni funkcije (jednadžba (2.9), slika 2.5b) te razne varijacije ([28], [33]).

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.9}$$



(a) ReLU aktivacijska funkcija.

(b) $\sigma(x)$ i $\tanh(x)$ funkcije

Slika 2.5: Varijante aktivacijske funkcije. Izvor: [46]

Dodavanjem aktivacijske funkcije, svaki sloj neuronske mreže predstavlja nelinearnu transformaciju. S parametrima koji se ažuriraju tijekom učenja, svaki sloj može se promatrati kao zaseban model koji uči na temelju izlaza prethodnog sloja i daje svoju reprezentaciju obrađenih podataka sljedećem sloju. Kako sljedeći sloj uči tu reprezentaciju, neuronske mreže su modeli koji uče reprezentacije podataka, po slojevima ([4]). Prosljeđivanje informacija po slojevima omogućuje sljedećim slojevima da grade na konceptima

prethodnih i treniraju se specifično za sloj apstrakcije koji odgovara njihovoj ulozi unutar mreže. Gradeći postupno slojeve apstrakcije, neuronske mreže u stanju su naučiti kompleksne koncepte poput prepoznavanja slika, učeći postupno po slojevima hijerarhijske razine apstrakcije, kako je opisano u poglavlju 1.3.

Ovakva arhitektura pogodna je za učenje mnogih kompleksnih koncepata. Međutim, nije toliko učinkovita u modeliranju nizovnih informacija, gdje sljedeći elementi ovise o prethodnima. Na primjer, za razumijevanje prirodnog jezika potrebno je zapamtiti riječi s početka rečenice kako bismo mogli odrediti kontekst i značenje riječi sve do kraja rečenice. Čitajući tekst, potrebno je držati informacije o prethodnim rečenicama, paragrafima, stranicama, kako bismo razumjeli ono što slijedi kasnije. Sve ovo upućuje na potrebu za pamćenjem informacija tijekom uzastopnih koraka učenja. Za to se koristi nešto izmijenjena arhitektura neuronskih mreža.

RNN mreže i LSTM jedinice

Ograničenje neuronskih mreža da ne pamte niti proslijeđuju informacije u prethodne slojeve može se zaobići kreiranjem povratnih veza (ciklusa) između neurona, koje će vraćati dio informacija natrag u isti ili prijašnje slojeve.

Mrežu koja implementira povratne veze zovemo povratna neuronska mreža (eng. *recurrent neural network - RNN*, [43]). Povratne neuronske mreže posebno su pogodne za probleme rada s nizovima ([28]) jer omogućuju čuvanje informacija o prijašnjim vremenskim koracima kroz buduće vremenske korake. Rečenice prirodnog jezika mogu se promatrati kao nizovi riječi gdje značenje pojedine riječi u rečenici ovisi o drugim riječima iste rečenice.

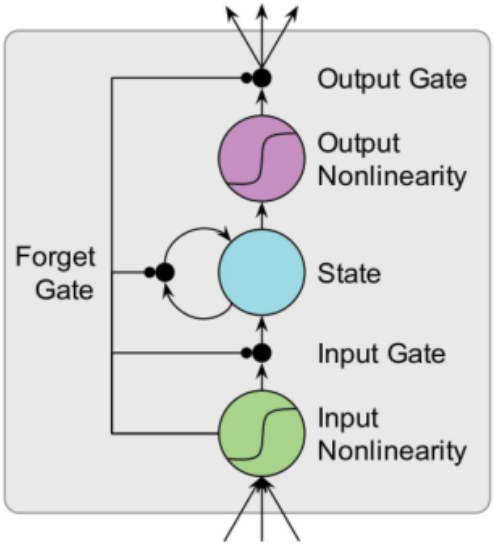
Jedan od koncepata koji omogućuje RNN mrežama dobro modeliranje sekvencijalnih podataka jest *dijeljenje parametara*. Neuroni s različitim parametrima različito obrađuju ulaze. No, pri obradi nizova, informacije se često ponavljaju - riječ može značiti isto bez obzira na položaj u rečenici, stoga treba naći način da se zabilježe isti podaci, bez potrebe da se treniraju u svakom neuronu ponovno. Dijeljenje parametara između različitih neurona omogućuje čuvanje sekvencijalnih informacija jer parametre ne učimo ispočetka u svakom vremenskom koraku ([28]). Također, time se postiže invarijantnost na redoslijed elemenata niza, jer će isti elementi niza dijeliti parametre, bez obzira gdje u nizu se nalaze.

Dijeljenje parametara tehnika je čuvanja nizovnih informacija kroz slojeve. Pored toga, čuvanje informacija moguće je i održavanjem unutarnjeg stanja pojedine jedinice u mreži. Ovakva jedinica znatno se razlikuje od prethodno predstavljenog neurona koji implementira jednosmjerno preslikavanje. Razvijena je stoga specifična arhitektura za ovakav tip zadatka, jedinice koje implementiraju unutarnje stanje, povratno proslijeđivanje informacija i dijeljenje parametara. Postoji više takvih, a najuspješnije su GRU ćelije ([14]) ili LSTM ćelije ([29]). Ovdje ćemo se više fokusirati na LSTM ćelije, jer su korištene u rado-

vima [41] i [11] te su pokazale velik uspjeh u ostalim zadacima obrade prirodnog jezika, pored prepoznavanja slijednosti (npr. strojno prevođenje [50], [3]).

LSTM ćelije (ćelije kratkoročno-dugoročne memorije, eng. *long-short term memory cell*, slika 2.6) sadrže povratne veze i unutarnje stanje što omogućuje pamćenje informacija tijekom više vremenskih koraka u odnosu na standardne neuronske mreže ([28]).

Osnovna karakteristika LSTM ćelija opisana je njihovim imenom: implementiraju kombinaciju kratkoročnog i dugoročnog pamćenja. LSTM sadrži memoriju u kojoj zadržava dio kontekstnih informacija pojedinog vremenskog koraka za zaključivanje o sljedećim vremenskim koracima. Pomoću nekoliko logičkih vrata za filtriranje informacija, u svakom vremenskom koraku algoritma učenja pojedina ćelija može zaboraviti ili zapamtiti informacije koje teku kroz nju, čime se formira kompleksnija slika koncepta koji se uči (slika 2.6). Kako svaka od logičkih vrata imaju svoju aktivacijsku funkciju, svaki od tih tokova informacija možemo smatrati vlastitom neuronskom mrežom koja uči koje informacije treba propustiti (u slučaju ulaznih vrata), zaboraviti (vrata zaborava) odnosno proslijediti dalje (izlazna vrata).



Slika 2.6: Stanje LSTM ćelije ovisi o ulazu, izlazu prethodnog koraka i stanju prethodne ćelije. Nelinearna aktivacijska funkcija je σ odnosno tanh, a protok podataka moduliran je pomoću troje vrata. Izvor: [2]

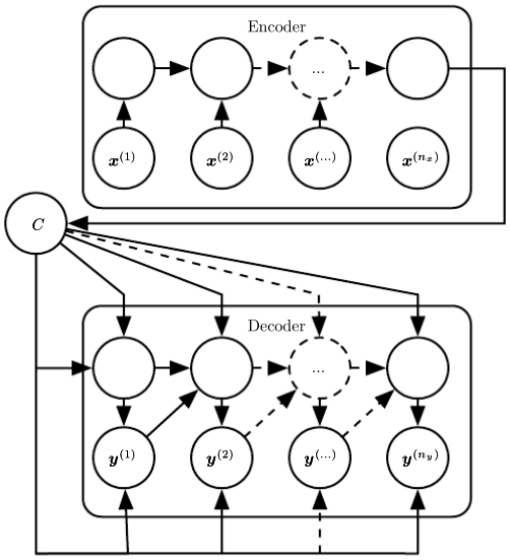
Na taj način, LSTM sloj, koji se sastoji od nekoliko povezanih LSTM ćelija, u neuronskoj mreži omogućuje dijeljenje parametara informacija koje prolaze kroz te ćelije. Ovaj mehanizam rezultira moćnom strukturom za čuvanje kontekstnih informacija dugačkih nizova, primjenjivom na širok spektar zadataka obrade nizovnih informacija. Na primjeru prepoznavanja slijednosti teksta, ulaz LSTM-a su vektorizirane reprezentacije riječi koje se proslijeđuju kroz nekoliko ulančanih LSTM ćelija. Svaka ćelija obrađuje jednu riječ i pamti dio informacija, zaboravlja dio informacija koje više nisu potrebne, a dio informacija proslijeđuje dalje. Tako se stvara sekvencijalna slika rečenice koju čitamo, uz očuvanje kontekstnih informacija svake riječi u skrivenom stanju LSTM-a.

Enkoder-dekoder RNN mreže

Neuronske mreže uče obradom ulančanih reprezentacija podataka. Svaki sloj, uključujući ulazni, proslijeđuje vlastitu obrađenu reprezentaciju ulaznih podataka sljedećem sloju. Međutim, svaki sloj ima fiksne dimenzije ulaza (slika 2.3). Pri radu s nizovima, postoji izazov njihove duljine - često je potrebno raditi s nizovima varijabilnih duljina. Na primjeru obrade rečenica, znatno ograničenje bi bilo kada bi model mogao obraditi rečenice samo točno određene duljine. Stoga je potrebno oblikovati mehanizam koji može kodirati ulaze varijabilnih duljina u njihovu reprezentaciju fiksne duljine.

Postupak reprezentacije niza podataka unaprijed nepoznate duljine vektorom fiksne duljine zovemo *enkodiranje* (eng. *encoding*). Pri preslikavanju niza varijabilne duljine u niz fiksne duljine potrebno je sačuvati nizovne informacije, o povezanosti elemenata niza. Potrebno je da rezultantni vektor bude dobra reprezentacija ulaznog niza podataka. Za postupak enkodiranja koriste se varijante RNN mreža upravo jer su one posebno pogodne za učenje nizovnih informacija i njihovu reprezentaciju.

Analogno, za neke zadatke postoji potreba transformacije vektora fiksne duljine (moguće prethodno enkodiranog) u niz varijabilne duljine. RNN mreže omogućuju i takvu transformaciju. Transformacija fiksnog vektora u niz zove se *dekodiranje* i slijedi sličan proces kao enkoder. Fiksni vektor biva proslijeđen nizu ulančanih ćelija RNN mreže, koje iz njega stvaraju nizovnu reprezentaciju fiksnog vektora koji im je proslijeđen. Jedna od primjena je automatsko opisivanje slika: generiranje opisa varijabilne duljine riječi iz slike koja je reprezentirana fiksnim vektorom ([53]).



Slika 2.7: Skica enkoder-dekoder modela. Najprije se vrši enkodiranje ulaznog niza **X** u kontekstni vektor **C**. Potom, **C** biva dekodiran u niz varijabilne duljine **Y**. Izvor: [28]

Kombinacija ova dva pristupa (slika 2.7) omogućuje konstrukciju tzv. enkoder-dekoder modela koji rade nad sekvencijalnim podacima i rezultiraju sekvencijalnim podacima. Primjena ovoga je brojna: strojno prevođenje mora funkcionirati i za rečenice izvornog teksta koje su različite duljine od rečenica rezultantnog jezika, automatsko odgovaranje na pitanja očekuje pitanja različite duljine i odgovore prilagođene pitanjima, a postoji i primjena na zadatku određivanja slijednosti. Modeli bazirani na enkoder-dekoder RNN mrežama prvi su postigli najbolje rezultate na tim područjima ([50], [14], [41], [11]).

Za dani niz $\mathbf{X} = (x^{(1)}, \ldots, x^{(n_x)})$, enkoder najprije prolazi nad nizom, akumulirajući informacije o pojedinim komponentama ulaza u svoja skrivena stanja te konačnu vrijednost sprema u tzv. *kontekstnu varijablu/vektor* $\mathbf{C}$. *Kontekst* $\mathbf{C}$ nakon enkodiranja može biti proslijeđen neuronskoj mreži na obradu ili direktno dekoderu. Dekoder preslikava kontekstni vektor u niz $\mathbf{Y} = (y^{(1)}, \ldots, y^{(n_y)})$ varijabilne duljine. Prednost enkoder-dekoder pristupa jest što može preslikati proizvoljne nizove različitih duljina n_x i n_y bez prethodnog znanja o odnosu duljina tih nizova.

Nedostak ovog pristupa jest ograničenje konteksta $\mathbf{C}$ fiksne duljine da reprezentira niz varijabilne duljine. Očito jest da pri preslikavanju nizova varijabilne duljine, ne mogu svi nizovi biti jednako dobro reprezentirani. Nadalje, kod enkoder-dekoder pristupa, dekoder nekad nema onoliko informacija o ulaznom nizu na temelju konteksta koliko bi imao da ima pristup ulaznom nizu ili izlaznim vektorima pojedinih vremenskih koraka enkodera. Neka od predloženih rješenja uključuju kontekst $\mathbf{C}$ kao niz varijabilne duljine ([3]) ili korištenje dodatne matrice, tzv. *neuralne pažnje* (eng. *neural attention*) koja služi za asociranje informacija o elementima ulaznog niza $\mathbf{X}$ s elementima izlaznog niza $\mathbf{Y}$, učeći elementarne informacije tijekom enkodiranja ([41]. [3]).

Neuralna pažnja

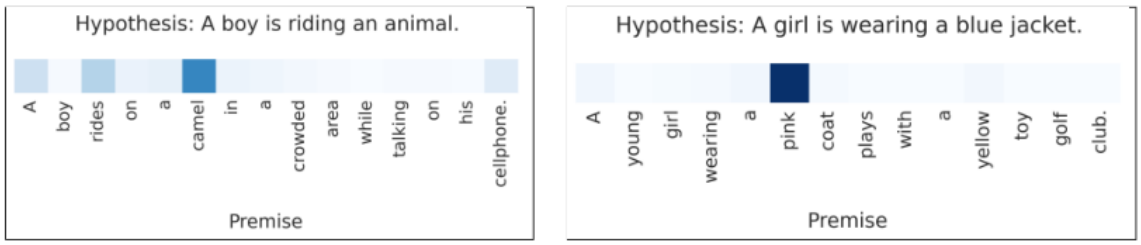
Neuralna pažnja ima za cilj riješiti nedostatke dekodera koji nema dovoljno nizovnih informacija za dekodiranje niza iz konteksta fiksne duljine. Kada bi uz kontekstni vektor, dekoder imao pristup pojedinim izlaznim vektorima skrivenih stanja enkodera, imao bi sekvencijalne informacije o svakom koraku enkodiranja što bi mu dalo više informacija o nizovnoj prirodi podataka u kontekstu.

Neuralna pažnja je matrica koja reprezentira odnose elemenata ulaznog niza s elementima izlaznog niza. Osnovna ideja jest u matricu zabilježiti informacije o svakom koraku rada enkodera, kako bi dekoder imao podatke o uzastopnim koracima prilikom reprezentacije niza kontekstom. Ovaj pristup ima za cilj skrenuti pažnju dekodera na relevantne dijelove ulaznog niza, odakle naziv neuralna pažnja.

Postoji nekoliko pristupa generiranja matrice pažnje. Moguće je kreirati težine koje predstavljaju važnost pojedinog izlaznog vektora vremenskog koraka enkodera ([41], slika 2.8) ili složenijim kreiranjem matrice čiji svaki element predstavlja usklađenost pojedinih elemenata enkodera i dekodera ([41], [39], [11]).

Općenito, korištenje pažnje pokazuje pozitivan učinak na uspješnost modela koji koriste enkoder-dekoder RNN mreže unutar svoje arhitekture. Sva tri modela koja obrađujemo u ovom radu ([41], [39], [11]) koriste neku varijantu neuralne pažnje te pokazuju bolje performanse korištenjem pažnje nego bez iste, a još bolje korištenjem pažnje na razini usklađivanja pojedinih elemenata enkodera i dekodera.

Osim na primjeni prepoznavanja slijednosti, neuronske mreže s ugrađenom pažnjom



(a) (b)

Slika 2.8: Vizualna reprezentacija pažnje na primjeru prepoznavanja slijednosti. Tamniji dijelovi reprezentiraju veće težine vektora pažnje, koje skreću pažnju dekodera na relevantne dijelove premise i hipoteze za zaključivanje o slijednosti. Izvor: [41]

u zadnjim godinama pokazale su se uspješnima u širokom spektru zadataka, od strojnog prevođenja ([3]), automatskog opisivanja slika riječima ([53]), sumarizacije teksta ([44]) pa do prepoznavanja govora ([16]).

Reprezentacija riječi

Pojašnjeno je kako neuronske mreže rade s reprezentacijama podataka, kao i problematika rada s ulaznim podacima različitih duljina (poglavlje 2.2). Pri određivanju slijednosti, potrebno je reprezentirati ulazne podatke na način pogodan za daljnju obradu. Moguće je reprezentirati riječi ([37], [40]), rečenice ([11]) ili čak čitave dokumente ([21]), različitih duljina, vektorima fiksne duljine, nad kojima potom vršimo klasifikaciju.

Optimalni rezultati za obradu prirodnog jezika postižu se upravo upotrebom unaprijed generiranih reprezentacija riječi (eng. *word embedding*) za najčešće korištene riječi. Ove reprezentacije oblikovane su s ciljem da semantički i sintaktički bliske riječi budu grupirane blizu u vektorskom prostoru u kojem su projicirane. Postoji nekoliko različitih reprezentacija riječi vektorima koje se koriste za zadatke procesiranja prirodnog jezika, od kojih su istaknute *word2vec* ([37]) i *GloVe* ([40]) reprezentacije.

Ove reprezentacije riječi pokazale su se uspješnima jer računske operacije nad enkodiranim vektorima čuvaju semantički kontekst riječi. Na primjer. oduzimanje enkodirane reprezentacije riječi *muškarac* od riječi *kralj* te dodavanjem vektora *žena* rezultira vektorom koji je bliži vektorskoj reprezentaciji riječi *kraljica* nego bilo kojem drugom vektoru u vektorskom prostoru enkodiranih vektora.

2.3 Metode učenja

Algoritam koji trenira neuronsku mrežu ažurira težine w i sklonosti b neurona u svakom koraku učenja tako da mreža kao funkcija f^* što bolje aproksimira ciljnu funkciju f. Parametri se prilagođavaju usmjeravanjem učenja prema izabranoj metrici. U poglavlju 2.1 kao jedna takva metrika naveden je iznos funkcije troška. Tako je problem učenja sveden na optimizacijski problem, koji ima za cilj pronaći minimum funkcije troška i ažurirati parametre na vrijednosti koje teže minimumu te funkcije za primjere u skupu za treniranje.

Stohastički gradijentni spust

Postoji nekoliko optimizacijskih algoritama za pronalazak minimuma funkcije troška. Jedan od najčešće korištenih za treniranje neuronskih mreža jest algoritam baziran na gradijentnom spustu - izračunu gradijenta funkcije troška (obzirom na parametre, jednadžba (2.10)) kako bi se dobio smjer gradijenta iz dane točke te ažurirali parametri u smjeru suprotnom od smjera gradijenta (prema lokalnom minimumu).

Ako označimo funkciju troška s C, a parametre modela (težine i sklonosti neurona) s $w_{jk}^{(L)}$ (težina j-og neurona sloja L za k-i ulaz prethodnog sloja) i $b_j^{(L)}$ (sklonost j-og neurona L-og sloja), gdje je n_L broj neurona L-og sloja, gradijent je dan jednadžbom 2.10

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w_{11}^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial b_1^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial w_{n_{L-1}n_L}^{(L)}} \\ \frac{\partial C}{\partial b_{n_L}^{(L)}} \end{bmatrix} \tag{2.10}$$

Računanje gradijenta funkcije nekoliko tisuća ili milijuna ([12], [11]) parametara vrlo je računalno intenzivno, a taj izračun potrebno je ponoviti više puta u više točaka (primjera iz trening skupa) kako bismo locirali minimum funkcije troška. Postoji nekoliko tehnika kojima se optimizira taj postupak.

Jedna mogućnost jest uzeti podskup trening primjera i računati gradijent funkcije samo nad tim primjerima, umjesto nad čitavim skupom primjera odjednom. Podskup može biti

nekoliko redova veličina manji od samog skupa pa nije uvijek reprezentivan za čitav skup. Međutim, birajući primjere nasumično i ponavljajući postupak mnogo puta nad više nasumično biranih skupina primjera, dobivamo reprezentativan smjer kretanja gradijenta u malim koracima koji je dovoljno dobar. Pošto je računanje gradijenta malog broja primjera znatno jednostavnije, ovaj postupak možemo ponoviti tisućama puta i time dobiti u prosjeku točan smjer kretanja gradijenta. Osim što zahtijeva manji broj računskih operacija, računanje nad malim nasumičnim uzorkom sigurnije će pronaći lokalne minimume za slučaj da je funkcija troška visoko zakrivljena i ima mnogo minimuma i točaka infleksije ([28], [5]).

Ovaj algoritam ne garantira da ćemo naći globalni minimum, no za mnoge primjene dovoljno je dobar i jedna je od najčešće korištenih tehnika računanja gradijenta. Računanje gradijenta kako bismo se "spuštali" niz hiperravninu malim koracima u potrazi za lokalnim minimumom zovemo gradijentni spust, a navedeni algoritam gdje optimiziramo potragu izborom nasumičnih primjera zovemo *stohastički gradijentni spust* (eng. *stohastic gradient descent, SGD*). Veličinu koraka zovemo *stopa učenja* (eng. *learning rate*).

Prolaskom kroz čitav skup ili dio njega, ažuriramo parametre. Dio skupa nad kojim računamo funkciju troška kako bismo ažurirali parametre u istom koraku učenja zovemo *(mini)serija* (eng. *(mini)batch*). Veličina ovog skupa hiperparametar je koji zovemo *veličina serije* (eng. *batch size*). Veći broj daje reprezentativniju sliku trening skupa u svakom koraku (što nije nužno bolje), ali je računalno intenzivniji. Trening skup particioniran je na takve serije te jedan prolazak kroz sve serije (sve primjere trening skupa) čini jednu *epohu* treniranja. Treniranje se obično vrši kroz više epoha (više prolazaka kroz trening skup) kako bi se dobile reprezentativne vrijednosti parametara za dani problem.

Propagacija greške unatrag

Kako bi istrenirao mrežu da uspješno modelira ciljnu funkciju, model treba ažurirati parametre neurona svih slojeva, uključujući skrivene slojeve. Za izlazni sloj ima pristup vrijednostima aktivacijske funkcije, ali nema izravno pristup međuvrijednostima skrivenih slojeva. Promatrajući izlaze i računajući greške izlaznih neurona, može se aproksimirati greška neurona u predzadnjem sloju. Ponavljajući postupak rekurzivno, može se procijeniti greška svakog od slojeva prije te sukladno ažurirati težine i sklonosti. Ovaj postupak propagacije greške u ranije slojeve zovemo *propagacija greške unatrag* (eng. *back-propagation, backprop* [43]). Taj algoritam u kombinaciji sa SGD predstavlja optimiziranu tehniku učenja neuronskih mreža.

Propagacija unatrag koristi *lančano svojstvo* (eng. *chain rule*) matematičke analize za računanje derivacija kompozicije funkcija. Svojstvo za vektore glasi: neka je $x \in \mathbb{R}^m, y \in$

$\mathbb{R}^n$, $g\colon \mathbb{R}^m \to \mathbb{R}^n$, $f\colon \mathbb{R}^n \to \mathbb{R}$. Ako vrijedi $y = g(x)$ i $z = f(g(x)) = f(y)$, onda:

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i} \tag{2.11}$$

Slojeve neuronske mreže promatramo kao funkcije, a način na koji su povezani, tako da si proslijeđuju rezultate računanja, kao kompoziciju funkcija (jednadžba (2.5), poglavlje 2.2). Stoga, za izračun gradijenta koji ovisi o parametrima zadnjeg sloja, možemo lančano "razmotati" ovisnosti o svim prethodnim slojevima kao u jednadžbi (2.11) budući da se radi o kompoziciji funkcija.

Funkciju cijene, koja računa udaljenost između rezultantne klase zadnjeg sloja i ispravne klase za pojedini primjer označit ćemo s C, a s $C^{(k)}$ funkciju greške koja računa grešku pojedinog primjera $k \in T$. Označimo s $a_j^{(L)}$ vrijednost aktivacijske funkcije (npr. σ) j-tog neurona sloja L (jednadžba (2.12)), a sa $z_j^{(L)}$ pokratu za težinsku sumu j-og neurona L-og sloja. Vidljivo je kako vrijednost aktivacije L-og sloja ovisi o vrijednosti aktivacije $(L-1)$-og sloja.

$$\begin{aligned} a_j^{(L)} &= \sigma\left(\sum_{k=0}^{n_{L-1}-1} w_{jk}^{(L)} a_k^{(L-1)} + b_j^{(L)}\right) \\ &= \sigma\left(z_j^{(L)}\right) \end{aligned} \tag{2.12}$$

Gradijent se računa obzirom na težine $w_{jk}^{(L)}$ i sklonosti $b_k^{(L)}$, čije vrijednosti pak ovise o aktivacijama $a_k^{(L-1)}$ prethodnog sloja. Parcijalne derivacije obzirom na vrijednost težina, sklonosti i aktivacija sloja L dane su redom jednadžbama (2.13), (2.14) i (2.15)).

$$\frac{\partial C}{\partial w_{jk}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}}\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}\frac{\partial C}{\partial a_j^{(L)}} \tag{2.13}$$

$$\frac{\partial C}{\partial b_j^{(L)}} = \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}}\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}\frac{\partial C}{\partial a_j^{(L)}} \tag{2.14}$$

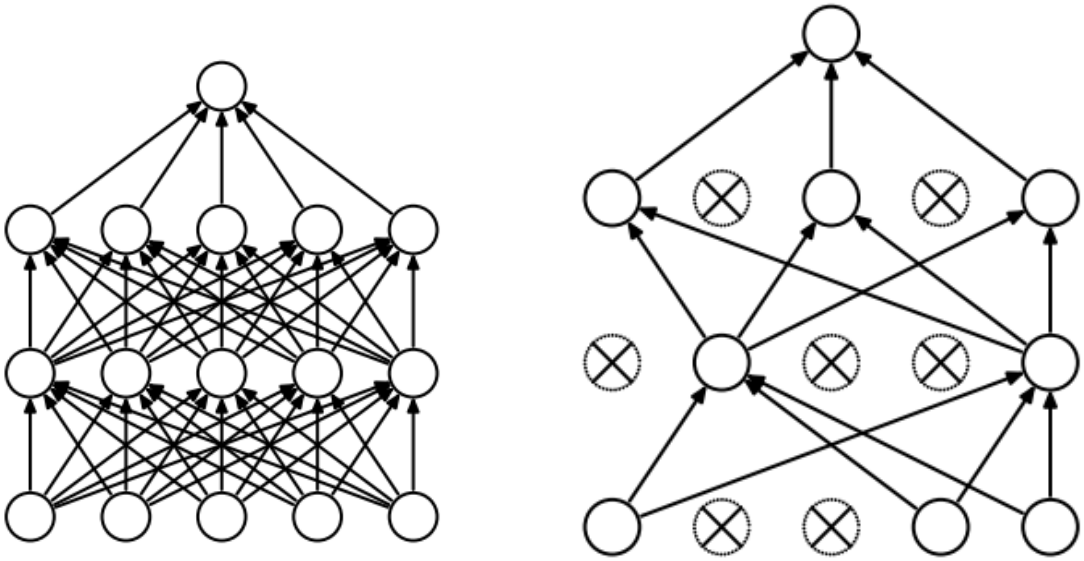
$$\frac{\partial C^{(k)}}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L-1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}}\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}\frac{\partial C^{(k)}}{\partial a_j^{(L)}} \tag{2.15}$$

Opisani postupak nam omogućuje da propagiramo grešku unatrag, tj. odredimo ovisnost greške posljednjeg sloja o parametrima prethodnih slojeva. Ulančano pravilo derivacija omogućuje da se vraćamo proizvoljno slojeva unatrag, ponavljajući postupak rekurzivno. Gradijent po definiciji sadrži sve parametre modela (jednadžba (2.10)), međutim obično se koristi neka tehnika optimizacije, poput SGD, koja smanjuje broj potrebnih računskih operacija.

2.4 Regularizacija i optimizacija modela

Prilikom treniranja modela gradijentnim spustom, korisno je implementirati dodatne tehnike optimizacije ili regularizacije modela. Optimizacija ima za cilj usmjeriti proces treniranja modela do optimalnih vrijednosti parametara obzirom na funkciju troška. To se postiže raznim tehnikama kao što su promjena brzine učenja ili optimizacija izračuna gradijenta za određivanje sljedećeg ažuriranja parametara. Regularizacija je skup tehnika koje imaju za cilj povećati generalizacijsku moć modela. Ovo se postiže penaliziranjem određenih svojstava funkcije modela ili uvođenjem šuma u podatke, kako bi se smanjila generalizacijska greška.

Regularizacija



(a) Potpuno povezana neuronska mreža.

(b) Neuronskoj mreži sa ispuštanjem nedostaju neke veze.

Slika 2.9: Razlika između potpuno povezane i djelomično povezane mreže (s ispuštanjem). Izvor: [48]

U sekciji 2.1 spomenuta je regularizacija kao tehnika sprečavanja pretreniranja modela na trening skupu. Pored $L1$ i $L2$ regularizacije, najčešće korištena metoda regularizacije neuronskih mreža je *ispuštanje*. Ispuštanje (eng. *dropout*, [48]) je računalno nezahtjevna tehnika regularizacije umjetnim uvođenjem šuma u podatke. Tehnika se sastoji u ispuštanju

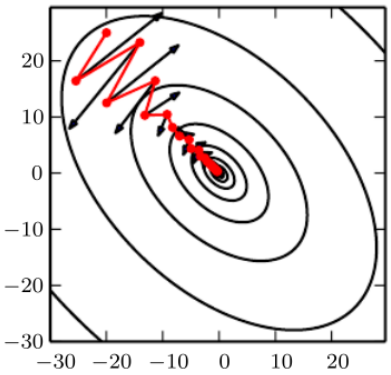
određenih neurona iz neuronske mreže, zanemarujući njihov izlaz u pojedinom koraku računanja, što se postiže množenjem izlaza tog neurona s 0 (slika 2.9b).

Koliko neurona biti ispušteno iz nekog vremenskog koraka (njihov izlaz zanemaren) određeno je hiperparametrom *stopa ispuštanja* (eng. *dropout rate*) koji je u intervalu $[0, 1]$ (obično 0.8 za ulazni sloj te 0.5 za skrivene slojeve [28]). Stopa ispuštanja određuje vjerojatnost da pojedini neuron u pojedinom vremenskom koraku bude nasumično ispušten. Ispuštanjem podskupa neurona iz ažuriranja parametara osigurava se da se model ne oslanja previše na pojedine neurone pri treniranju (izbjegavanje pretreniranja).

Ispuštanje dobro skalira s povećanjem skupa podataka te poboljšava generalizaciju modela, odnosno smanjuje generalizacijsku grešku nad neviđenim primjerima ([48]).

Optimizacija učenja

Neprekinutim treniranjem može doći do pretreniranja modela na dani skup podataka. Stoga je potrebno prekinuti treniranje kada model stagnira po metrici po kojoj se usmjerava učenje (npr. iznos funkcije troška). Uvjet pod kojim želimo zaustaviti učenje modela zovemo *kriterij ranijeg zaustavljanja* (eng. *early stopping*). Standardni kriteriji zaustavljanja mogu biti dosezanje platoa funkcije troška, fiksni broj epoha ažuriranja parametara ili prevelika razlika mjerene greške između trening i validacijskog skupa (što može značiti rast generalizacijske greške).



Slika 2.10: Vizualizacija zamaha: crne strelice pokazuju smjer gradijenta u svakom koraku učenja, a crveno je smjer kretanja učenja koje koristi zamah - otpornije je na varijacije u gradijentu kako učenje napreduje. Izvor: [28]

Jedna od tehnika optimizcije učenja računanjem gradijenta jest korištenjem tzv. *zamaha* (eng. *momentum*) (slika 2.10). Poput pojave u fizici, zamah daje određenu inerciju algoritmu učenja koji je stekao brzinu u nekom smjeru tako da je smjer učenja manje podložan promjenama kako učenje napreduje (kao što tijelo ima inerciju pri kretanju). Stopa učenja se smanjuje pa velike devijacije u smjeru gradijenta sve manje utječu na oblikovanje parametara.

Općenito, korisno je prilagođavati brzinu učenja da ona ne bude konstanta tijekom cijelog procesa jer se na taj način postiže robusnost modela na šum u podacima ([22], [28]). Posebno kod SGD, moguće je naići na nekoliko rubnih primjera u seriji koji odskaču karakteristikama. Takvi primjer navodit će kretanje učenja u krivom smjeru, a smanjenjem brzine učenja nakon početnih iteracija osigurava se da ovakvi primjeri imaju manji učinak na rezultat.

Postoji nekoliko pristupa kako optimizirati smjer i brzinu kretanja koraka učenja prilikom spuštanja niz gradijent. Adagrad ([25]), RMSprop ([42]) i Adam ([34]) neki su od optimizacijskih algoritama, a uključuju prilagodbe kao što su promjena stope učenja različito nad različitim parametrima, promjena iznosa zamaha, brzine učenja ili "rezanja" koraka u smjeru kretanja gradijenta (skraćivanje gradijenta, eng. *gradient clipping*, [28]). Takve metode poboljšavaju robusnost modela baziranih na stohastičkom gradijentnom spustu ([22]) i poboljšavaju uspješnost modela u odnosu na standardni gradijentni spust bez optimizacija.

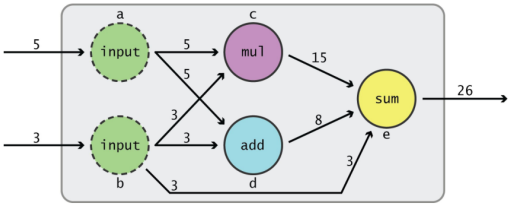
Neke tehnike optimizacije mogu biti korištene zajedno (npr. stohastički gradijentni spust kao tehnika računanja gradijenta uz Adam optimizator kao optimizaciju ažuriranja brzine učenja pri stohastičkom spustu niz gradijent) i kraćenje gradijenta. Također, moguće je kombinirati tehnike optimizacije i regularizacije u sinergiji za bolje rezultate modela. Sve to su oblikovne odluke pri izgradnji modela dubokog učenja koje rezultiraju velikim manevarskim prostorom za izgradnju kvalitetnog modela.

Optimizacija hiperparametara

Čak i za fiksnu arhitekturu modela, postoje mnogi hiperparametri koje je potrebno podesiti kao što su stopa učenja, veličina serije, faktor ispuštanja kod ispuštanja, hiperparametri optimizacijskih funkcija i sl. Svaki od ovih faktora utječe na krajnji rezultat pa je za pronalazak optimalne konfiguracije hiperparametara potrebno isprobati više njhovih kombinacija. No, zbog velikog broja parametara i složenosti modela, evaluacija modela jedne konfiguracije može trajati satima ili danima. Stoga je cilj minimizirati broj evaluacija modela.

Postoje razni pristupi iskušavanju kombinacija hiperparametara. Parametre promatramo kao raspoređene u mrežu mogućih vrijednosti, poput tablice, te se vrši pretraživanje prostora tih vrijednosti. Ako je kombnacija malo ili evaluacija modela nije vremenski skupa, dovoljno je jednostavno pretražiti sve kombinacije kroz mrežu (eng. *grid search*), ručno pretraživati pojedine konfiguracije ili nasumično pretraživati (eng. *random search*). Nasumično pretraživanje pokazalo se učinkovitijim od ručnog ili potpunog pretraživanja mreže ([7]). Međutim, barem jednako učinkovitim pokazuje se vjerojatnosna optimizacija koja u obzir uzima rezultate prethodnih evaluacija - Bayesova optimizacija.

Bayesova optimizacija hiperparametara bazira se na izgradnji vjerojatnosnog modela tzv. surogat funkcije koja preslikava kombinacije hiperparametara u vrijednost ciljne funkcije modela. Ciljna funkcija je obično funkcija greške ili točnosti modela, evaluirana na validacijskom skupu. Evaluirajući modele različitih konfiguracija hiperparametara te njihovu uspješnost, Bayesova optimizacija skuplja informacije o distribuciji surogat funkcije te traži njen optimum. Ovaj pristup kombinira isprobavanje nepoznatih konfiguracija hiperparametara s optimiziranjem konfiguracija koje su se pokazale uspješnima kako bi u što manje evaluacija modela pronašla konfiguraciju hiperparametara koja postiže najbolje rezultate.



(a) Primjer grafa u TensorFlowu: podaci putuju duž usmjerenih bridova. Funkcije množenja *mul* i zbrajanja *add* (čvorovi) transformiraju podatke.

(b) Čitav graf je jedna transformacija.

Slika 2.11: TensorFlow model računanja kao usmjereni aciklički graf. Izvor: [2]

Odabir sljedeće konfiguracije hiperparametara za evaluaciju ovisi o vjerojatnosnom modelu koji uzima u obzir prethodne evaluacije i njihovu uspješnost. Primjeri takvih modela su Gaussovi procesi (eng. *Gaussian process* - *GP* [47], [6]) ili stablasto strukturirani Parzen procjenitelj (eng. *tree-structured Parzen estimator* - *TPE* [6]). U oba slučaja, Bayesova optimizacija pokazala se uspješnijom od pretraživanja mreže i nasumičnog pretraživanja ([6], [47], [32]). U ovom radu koristit ćemo Bayesovu optimizaciju hiperparametara s TPE modelom u pozadini.

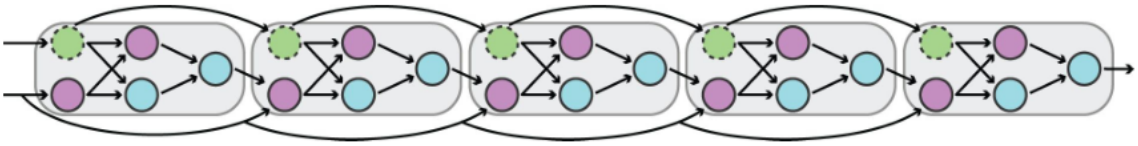
2.5 Implementacija

Tensorflow

TensorFlow ([1]) je Python biblioteka otvorenog koda (eng. *open source*) razvijena od strane Google-a, namijenjena za numeričke izračune visokih performansi. Dolazi s podrškom za strojno i duboko učenje kroz sučelje u Python programskom jeziku koje olakšava konstrukciju modela.

Osnovni računski model TensorFlowa reprezentiran je usmjerenim acikličkim grafom (slika 2.11a). Čvorovi tog grafa predstavljaju funkcijska preslikavanja (računske operacije), a bridovi označavaju podatke koji se proslijeđuju između čvorova (skalari, matrice, tenzori). Na ovaj način stvorena je intuitivno razumljiva podloga za izgradnju modela strojnog učenja jer se već modeli strojnog/dubokog učenja oblikuju kao grafovi računskih operacija. Ovom arhitekturom moguće je kreirati proizvoljno složene konstrukcije od elementarnih. Osim toga, distribucijom operacija u obliku grafa omogućuje se paralelizacija na prikladnom hardveru jer se modeli dubokog učenja često sastoje od velikog broja računalno jednostavnih operacija koji se mogu izvršavati u paraleli.

Kompozicijom jednostavnijih dobivaju se složene operacije. Možemo povezivati računske grafove u veće grafove te tretirati pojedine grafove kao čvorove jer je čitav graf i



Slika 2.12: Razmotavanjem grafa izbjegavaju se ciklusi, a dobiva željeni rezultat. Izvor: [2]

sâm funkcijsko preslikavanje, kao kompozicija unutarnjih funkcijskih preslikavanja (slika 2.11b). Za pojedine arhitekture neuronskih mreža potrebno je stvoriti povratne veze među računskim čvorovima (npr. povratne neuronske mreže, RNN). Ciklusi u grafu nisu dozvoljeni jer evaluacija računskih operacija ide u smjeru bridova, a postojanjem ciklusa evaluacija, pa time ni izvršavanje programa, nikada ne bi završilo. Stoga se implementira "razmotavanje" kojim se ciklusi pretvaraju u sekvencijalne veze između uzastopnih grafova (slika 2.12).

To je osnovna filozofija konstrukcije modela u TensorFlowu. TensorFlow implementira unaprijed definirane transformacije podataka koje predstavljaju razne komponente modela strojnog učenja, odnosno slojeve neuronskih mreža. Pokretanje modela svodi se na dva koraka:

1. Definiranje računskog grafa
2. Izvršavanje grafa (s podacima)

Izvršavanje grafa s podacima znači davanje ulaznih podataka modelu. Kada su ulazni čvorovi grafa dobili podatke, izvršavanjem grafa započinje se njime definiran postupak izračuna.

Keras

Keras ([15]) je API (eng. *application programming interface*) - programsko sučelje za rad s neuronskim mrežama, napisan u Pythonu, koji može raditi kao sloj apstrakcije nad TensorFlowom ili Theanom ([51]). U ovom radu koristit ćemo ga u kombinaciji s TensorFlowom.

Keras slijedi paradigmu objektno orijentiranog programiranja te nudi komponente neuronskih mreža kao gotove objekte u programskom jeziku Python. Sadrži implementacije svih najčešće korištenih komponenti za izgradnju modela strojnog/dubokog učenja po slojevima korištenjem objektno orijentirane sintakse, uz podršku TensorFlow implementacije u pozadini. Stvaranje modela u Kerasu stoga je pojednostavljeno na nekoliko linija koda. Primjer stvaranja modela i dodavanja dva sloja, s definiranjem aktivacijske funkcije i vrste klasifikatora:

```
model = Sequential()
model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))
```

S druge strane, u TensorFlowu bi novi sloj morao biti dodan definiranjem odgovarajućih računskih grafova, njihovim komponiranjem i povezivanjem. To omogućuje izgradnju i pokretanje modela u Kerasu u puno kraćem vremenu, a njihova izmjena i razumijevanje jednostavni su zbog pristupačnijeg korisničkog sučelja u odnosu na TensorFlow. Budući da u pozadini koristi TensorFlow za evaluaciju računskih izraza, kod u Kerasu je usporedivo brz i podržava paralelizaciju na posebnom hardveru, kao i TensorFlow.

Poglavlje 3

Radovi

U ovom poglavlju radimo pregled nekoliko klasifikacijskih modela za rješavanje problema slijednosti teksta, baziranih na dubokom učenju ([39], [41], [11]). Najprije opisujemo općenitu strukturu ovakvih modela i ukratko objašnjavamo uloge pojedinih komponenti. Potom pojedinačno razlažemo modele svakog od radova ističući implementacijske razlike u odnosu na osnovnu strukturu. Naposljetku spominjemo i međusobno uspoređujemo rezultate koje postiže svaki model.

3.1 Općenita struktura modela

Reprezentacija ulaza. Za reprezentaciju riječi iz rečenica koristi se neka od dvije trenutno najuspješnije implementacije unaprijed treniranih reprezentacija riječi (GloVe [40] ili word2vec [37]). Premisa **X** i hipoteza **Y** tako su reprezentirane varijabilnim nizovima vektora koji predstavljaju kodiranja pojedinih riječi.

Sloj za sekvencijalno kodiranje. Nad takvim podacima djeluje sekvencijalni enkodirajući sloj u obliku povratne neuronske mreže, koja stvara rečenične reprezentacije kako bi model naučio semantičko značenje rečenica koje nije zabilježeno enkodiranjem riječi kao diskretnih vektora. Enkodiranjem rečenica različitih duljina u kontekst fiksne duljine gubi se dio informacija o pojedinim dijelovima rečenica. Posebno, izostaju informacije o relevantnosti pojedinih fraza unutar konteksta rečenice za zaključivanje o slijednosti. Pokazano je kako veću uspješnost u obradi prirodnog jezika imamo kada imamo informacije o *usklađenosti relevantnih dijelova rečenica* koje obrađujemo (npr. strojno prevođenje [50]).

Neuralna pažnja. Sloj neuralne pažnje simulira pažnju (pozornost) modela stvarajući vektor težina koji ima za cilj naučiti težine koje opisuju koje fraze premise su relevantne za koji dio hipoteze. Ovo ima za cilj riješiti prethodno navedeni nedostatak konteksta pojedinih riječi unutar rečenice. Neuralna pažnja djeluje kako je radi kako je zamišljeno, ističući povezane dijelove premise i hipoteze modelu za bolje zaključivanje o slijednosti.

Klasifikator. Nakon sekvencijalnih obrada, koje imaju za cilj naučiti što više relevantnih informacija o odnosu dvaju rečenica, njihove vektorske reprezentacije sa naučenim semantičkim značenjem konkateniraju se u rezultantnu reprezentaciju para koja potom biva proslijeđena klasifikatoru. To je obično neka varijanta povratne ili unaprijedne višeslojne neuronske mreže koja skupljene informacije klasificira u jednu od klasa kojoj pripada par rečenica (slijednost, kontradikcija, neutralan odnos).

Metrika i skup podataka. U svakom eksperimentu kao mjera uspješnosti koristi se točnost (eng. *accuracy*). Treniranje i testiranje vrši se nad SNLI skupom podataka ([10]). Ako nije drugačije navedeno, izuzeti su parovi za koje ne postoji konsenzus, a za validaciju i testiranje korišteni su unaprijed konstruirani testni i validacijski skupovi od po 9842 primjera distribuirani uz SNLI. Točnost modela mjeri se rezultatom nad testnim skupom. Modeli se uspoređuju međusobno, kao i sa relevantnim modelima u vrijeme pisanja rada u kojem su predstavljeni.

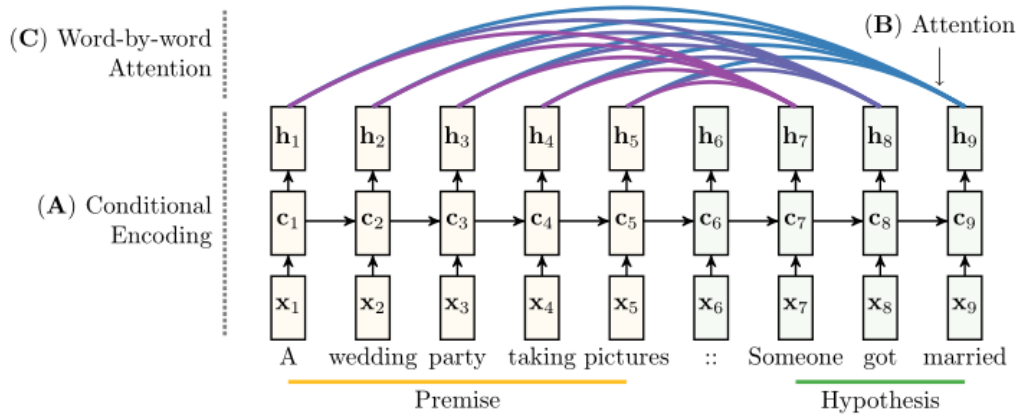
3.2 Zaključivanje o slijednosti korištenjem pažnje

Model za određivanje slijednosti baziran na neuralnoj pažnji iz [41] određuje slijednost čitajući premisu i hipotezu zajedno, prolazeći nad parovima riječi i izraza u premisi i hipotezi istovremeno. Model usklađuje fraze premise i hipoteze koji međusobno odgovaraju pomoću neuralne pažnje.

Pregled modela

Reprezentacija ulaza. Za reprezentacije riječi koriste se unaprijed enkodirani *word2vec* vektori čije se vrijednosti *ne prilagođavaju* za vrijeme treninga. Riječi izvan vokabulara na skupu za treniranje inicijaliziraju se uniformnom distribucijom nasumičnim vrijednostima iz intervala $[-0.05, 0.05]$ te se (te vrijednosti) prilagođavaju za vrijeme treninga. Riječi izvan vokabulara na koje naiđemo u validacijskom i testnom skupu inicijaliziraju se na fiksne vektore nasumičnih vrijednosti. Vektore riječi iz vokabulara ne ažuriramo za vrijeme treninga kako bi ti vektori bili bliži vektorima njima bliskih riječi ili izraza na koje ćemo naići prilikom određivanja slijednosti u hipotezi, jer znamo da su vektori sličnog semantičkog konteksta međusobno bliži u vektorskom prostoru word2vec reprezentacija ([37]).

Sloj za sekvencijalno kodiranje. Ulaz modelu su unaprijed enkodirane reprezentacije riječi word2vec vektorima ([37]). Sekvencijalni enkodirajući sloj implementiran je nizom LSTM ćelija koje kodiraju nizove riječi (slika 3.1). Sastoji se od dva LSTM-a, po jedan za premisu i hipotezu. Ulaz LSTM ćelije c_t su vektor x_t kao reprezentacija riječi i stanje prethodnog LSTM-a c_{t-1}, a izlaz je vektor h_t za vremenski korak t. Nakon što prvi LSTM pročita premisu, drugi LSTM čita hipotezu, s tim da je prva ćelija drugog LSTM-a inicijalizirana stanjem posljednje ćelije prvog LSTM-a.



Slika 3.1: Obrade premise i hipoteze pomoću LSTM-a. Izvor: [41]

Neuralna pažnja. Na ovaj način sekvencijalno se uče informacije iz premise i hipoteze zajedno. Obzirom da se dio informacija o početku premise u LSTM jedinicama može izgubiti do dalekog vremenskog koraka, implementira se varijanta neuralne pažnje. Pažnja je implementirana kao vektor težina koje ukazuju koje riječi premise su najrelevantnije za danu hipotezu. Ovo je varijanta mekog usklađivanja fraza premise i hipoteze koja je pokazala uspjeh u dosadašnjim modelima ([3], [44]).

Dodatno, varijanta modela koristi **pažnju na razini riječi** (eng. *word based attention*) na sljedeći način: drugi LSTM, čitajući svaku riječ hipoteze, prolazi nad svim izlaznim vektorima h_t (riječima) prvog LSTM-a i svaki od njih uspoređuje s trenutnom rječju hipoteze. Stvara se matrica dimenzija |br.riječi premise| $\times$ |br.riječi hipoteze| u kojoj su težine koje predstavljaju povezanost svake riječi premise i hipoteze. Ova matrica pomaže modelu da nauči koje riječi dvaju rečenica su međusobno najrelevantnije.

Klasifikator. Klasifikator je sloj potpuno povezanih perceptrona sa tangens hiperbolni aktivacijskom funkcijom i *softmax* klasifikatorom.

Hiperparametri i rezultati

Za optimizaciju korišten je optimizator Adam sa standardnim parametrima navedenima u radu [34]. Funkcija troška je funkcija unakrsne entropije. Za svaki model, koristi se pretraživanje mreže svih vrijednosti hiperparametara kao metoda optimizacije hiperparametara:

- stopa učenja iz skupa $\left\{10^{-4}, 3 \cdot 10^{-4}, 10^{-3}\right\}$
- stopa ispuštanja iz skupa $\{0.0, 0.1, 0.2\}$
- te $L2$-regularizacijski koeficijent iz skupa $\left\{0.0, 10^{-4}, 3 \cdot 10^{-4}, 10^{-3}\right\}$,

MODEL	TRAIN	DEV	TEST	#PARAM.
LSTM [10]	84.4	-	77.6	221k
Klasifikator [10]	99.7	-	78.2	-
LSTM [41]	83.3	82.1	80.9	252k
LSTM + Pažnja [41]	85.4	83.2	82.3	242k
LSTM + Pažnja (dvosmjerna) [41]	86.5	83.0	82.4	242k
pažnja na razini riječi [41]	85.3	**83.7**	**83.5**	252k
pažnja na razini riječi (obostrana) [41]	86.6	83.6	83.2	252k

Tablica 3.1: Rezultati modela s navedenim točnostima i brojem parametara za usporedbu

mjeri se uspješnost najbolje kombinacije nad validacijskim skupom i *samo se ta* kombinacija hiperparametara koristi za mjerenje uspješnosti modela nad testnim skupom. Najpovoljnija kombinacija hiperparametara daje rezultate vidljive u tablici 3.1.

Ovaj model predstavlja prvo rješenje bazirano na dubokom učenju koje postiže performanse bolje od klasifikatora s ručno biranim značajkama za RTE zadatak. Dotadašnji modeli bazirali su se na klasičnom pristupu, što je vidljivo na klasifikatoru [10] koji klasifikaciju bazira na značajkama rečenica kao što su

- usporedba duljina rečenica
- preklapanje riječi u rečenicama
- prepoznavanje sličnosti na razini bi- i uni-grama

i sličnima. Uspjeh dubokih modela može se zahvaliti kvalitetnom skupu za treniranje koji omogućuje primjenu modela dubokog učenja koji su se dosad pokazali efikasnima za NLP zadatke.

Za razliku od LSTM implementacije [10], koja na testnom skupu postiže 0.6% lošije rezultate nego klasifikator s ručno biranim značajkama, ova LSTM implementacija postiže još veću točnost od **83.5%**. LSTM jedinice omogućuju dugoročno čuvanje informacija o pročitanom tekstu pa se relevantne informacije ne gube kao kod sekvencijalnog čitanja, nego bivaju propagirane dalje, a neuralna pažnja omogućuje meko usklađivanje relevantnih dijelova premise i hipoteze, što daje poboljšanje od 1.4% u odnosu na samo LSTM-ove za kodiranje rečenica.

3.3 Razloživi model s pažnjom

Modeli s LSTM ćelijama ([41], [10]) koriste kompleksnu arhitekturu kako bi zabilježili sekvencijalnu prirodu rečenica enkodirajući ih prije njihove klasifikacije. Korištenje paž-

nje u kombinaciji s LSTM-om u ovim kompleksnim modelima rezultira velikim brojem parametara, što povećava računalnu kompleksnost modela.

Premisa ovog eksperimenta jest da je dovoljno promatrati pojedine dijelove odgovarajućih tekstova (premise i hipoteze), usporediti relevantne dijelove pomoću pažnje te na temelju toga donijeti zaključak o slijednosti tekstova. Time problem određivanja slijednosti razlažemo na više manjih problema uspoređivanja podfraza i riječi, pa je ovaj model nazvan "Razloživi model s pažnjom" (eng. *decomposable attention model*.

Pregled modela

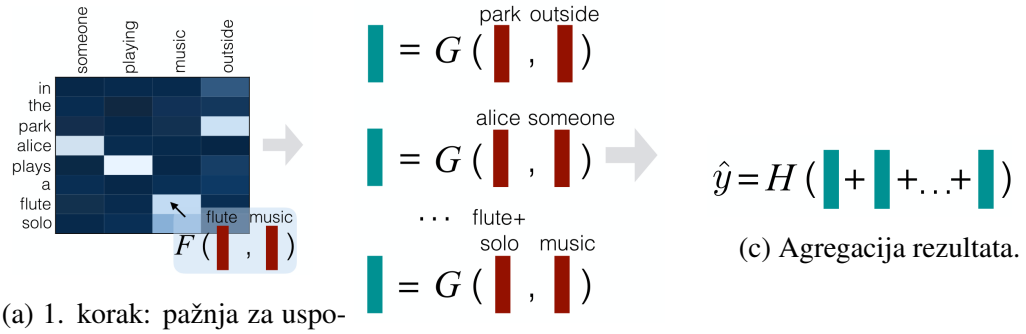
Reprezentacija ulaza. Kao ulaz modelu služe GloVe vektorske reprezentacije riječi ([40]). Pošto nema sloja za sekvencijalnu reprezentaciju, model ne barata nizovima nego radi direktno nad enkodiranim riječima. Varijanta modela koristi unutar rečeničnu pažnju kako bi promijenila ulaznu reprezentaciju rečenica tako da enkodiranje rečenice ipak sadrži neke sekvencijalne informacije. Ovo enkodiranje se vrši sumiranjem i usrednjivanjem svake rečenice, što biva proslijeđeno unaprijednoj neuronskoj mreži. Dobivena enkodiranja konkateniraju se na početne vektorske reprezentacije (bazirane na GloVe projekcijama) i proslijeđuju se dalje modelu.

Sloj za sekvencijalno kodiranje ovdje nije prisutan. Pošto ovaj model razlaže problem određivanja slijednosti na probleme slaganja podfraza rečenica, korak enkodiranja sekvencijalnih informacija izostavljen je kako bi se zaključivalo direktno nad parovima fraza premise i hipoteze. Umjesto toga, ovaj model razložen je na komponente *pažnje*, *uspoređivanja rečenica* i *agregacije* rezultata dotadašnjih slojeva što se proslijeđuje klasifikatoru.

Neuralna pažnja. Model čita riječi premise i hipoteze kako bi odredio odnose pojedinih riječi između rečenica. Svaku riječ premise kombinira sa svakom rječju hipoteze kako bi našao podizraze hipoteze koji semantički najviše odgovaraju pojedinoj riječi premise, i obratno. Pošto nisu sve kombinacije riječi dvaju rečenica jednako relevantne, računaju se *težine* koje govore o toj relevantnosti proslijeđujući vektorske reprezentacije rečenica unaprijednoj MLP mreži s ReLU aktivacijama ([27]).

Na taj način *usklađujemo* relevantne dijelove dvaju tekstova, što nam omogućuje da kasnije za zaključak o slijednosti više poštujemo odnose relevantnih dijelova rečenica u odnosu na irelevantne (slika 3.2a). Očito jest da se manje pozornosti obraća na irelevantne kombinacije riječi (prilozi poput "a", "in", "the" ne govore nam ništa o semantičkoj strukturi). Izračunate težine svake rečenice potom normaliziramo obzirom na sve težine te rečenice. Normalizirane težine predstavljaju "usklađene" dijelove rečenica.

Uspoređivanje. Sada za svaku frazu jedne rečenice imamo informaciju o tome koji dio druge rečenice joj je semantički najbliži. Uspoređujemo te informacije u oba smjera (slika 3.2b) unaprijednom neuronskom mrežom kao u prethodnom koraku. Rezultat su vektori,



(a) 1. korak: pažnja za usporedbu relevantnih dijelova rečenice.

(b) 2. korak: usporedba fraza dvaju rečenica.

(c) Agregacija rezultata.

Slika 3.2: Razloživi model u 3 koraka. Izvor: [39]

kojih za svaku rečenicu ima koliko ima riječi u njoj, a koji sadrže nelinearne kombinacije fraza iz premise i njima odgovarajućih fraza iz hipoteze (i obratno).

Agregacija i klasifikator. Naposljetku, agregiramo rezultate za svaku rečenicu zbrajajući usklađene vektore izraza. Dobivene sume konkateniramo te prosljeđujemo unaprijednoj neuronskoj mreži (slika 3.2c) nakon koje slijedi linearni *argmax* klasifikacijski sloj.

Hiperparametri i rezultati

Za reprezentaciju riječi koriste se predtrenirane 300-dimenzionalne GloVe reprezentacije projicirane na 200 dimenzija ([40]) koje su normalizirane na $L2$ normu od 1. Riječi izvan vokabulara reprezentirane su nekom od 100 nasumično inicijaliziranih reprezentacija, svaka od kojih je inicijalizirana srednjom vrijednošću 0 i standardnom devijacijom od 1. Enkodiranja riječi *ne mijenjamo* tijekom treninga kako bi reprezentacije ostale što sličnijima neviđenim riječima u testnom skupu. Ostale težine mreže inicijalizirane su na nasumične vrijednosti s normalnom distribucijom srednje vrijednosti 0 i standardne devijacije 0.01.

Za optimizaciju je korišten Adagrad optimizator s inicijalnim hiperparametrom akumuliranja 0.1. Za regularizaciju korištena je tehnika ispuštanja ([48]) s faktorom 0.2 u svakom, osim zadnjeg linearnog sloja (gdje nije korištena). Sve spomenute mreže imaju po 2 sloja s po 200 neurona. Inicijalna brzina učenja jest 0.05 za standardni, odnosno 0.025 za model s unutar-rečeničnom pažnjom. Funkcija cijene je funkcija unakrsne entropije (eng. *cross-entropy loss*).

U tablici 3.2 navodimo dva modela iz [10]: klasični klasifikator s ručno selektiranim značajkama, LSTM klasifikator iz istog rada, LSTM klasifikator s neuralnom pažnjom na razini riječi iz 3.2 ([41]), model iz ([12]) koji je u vrijeme predstavljanja modela bio

MODEL	TRAIN	TEST	#PARAM
LSTM [10]	84.4	77.6	221k
Klasifikator [10]	99.7	78.2	-
100D LSTM s pažnjom na razini riječi [41]	85.3	83.5	252k
450D LSTMN s pažnjom [12]	88.5	86.3	3.4M
Razloživi model [39]	89.5	86.3	382k
Razloživi model s unutar-rečeničnom pažnjom [39]	90.5	**86.8**	582k

Tablica 3.2: Rezultati - usporedba uspješnosti i broja parametaras s drugim relevantnim modelima

najuspješniji model za RTE zadatak te dvije varijante ovdje opisanog modela (sa i bez unutar-rečenične pažnje). Broj parametara je naveden bez parametara koji se odnose na enkodiranja riječi.

Ovdje predstavljen razloživi model postiže najbolje performanse, nadmašivši time i mnogo kompleksnije modele s mnogostruko više parametara. Dodavanje unutar-rečenične pažnje rezultira neznatnim poboljšanjem u odnosu na standardni model.

Razloživi model predstavlja poboljšanje i u odnosu na LSTM s međurečeničnom pažnjom (poglavlje 3.2, [41]). Vidjeli smo kako taj model implementira mnoge kompleksne metode s ciljem stjecanja što više razine razumijevanja konteksta, uključujući korištenje LSTM jedinica koje pamte relevantne informacije, međurečeničnu pažnju te pažnju na razini riječi, svaka od kojih u sinergiji s ostalima poboljšava uspješnost tog modela. Ovdje smo se fokusirali na izoliranje relevantnih podfraza premise i hipoteze te njihovu usporedbu.

Ne samo da ovaj model postiže najbolji rezultat dosad opserviran nad SNLI skupom podataka, nego to čini uz znatne uštede u performansama. Velik naglasak kod oblikovanja ovog modela bio je na konstrukciji računalno efikasnog algoritma za prepoznavanje slijednosti tekstova uz vrhunske performanse. Uz činjenicu da se vektori reprezentacije riječi ne treniraju, ukupan broj trenirabilnih parametara ovog modela jest 382k, što je za red veličine manje od drugog najuspješnijeg modela iz [12], koji ima 3.4M parametara.

Ovaj rezultat nam govori kako rješenja fokusirana na pojedine aspekte RTE problema mogu rezultirati visokom uspješnošću čak i bez duboke semantičke obrade od strane kompleksnih arhitektura za duboko učenje. Premda nisu rađeni statistički testovi koji bi usporedili značajnost ovih rezultata s dosadašnjim modelima, postizanje najboljih rezultata u usporedbi s mnogo složenijim modelima veliki je uspjeh obzirom na malen broj parametara i jednostavniji pristup rješavanju problema slijednosti.

3.4 Napredni sekvencijalni LSTM model za određivanje slijednosti (ESIM)

Ovdje predstavljeni rad ([11]) implementira nekoliko sekvencijalnih enkodiranja u različitim stadijima obrade podataka kao potencijalan alat za konstrukciju uspješnog modela prepoznavanja slijednosti teksta. U kombinaciji s neuralnom pažnjom, konstruirani model postiže dosad najveću zabilježenu točnost. Rad nudi i nadogradnju izgrađenog modela koristeći parsirane informacije o rečenicama i sintaksna stabla za zaključivanje o sintaktičkoj strukturi. Premda se s tom nadogradnjom postiže poboljšanje na testnom skupu, tu varijantu nećemo obraditi detaljno jer nudi mnogo kompliciraniji model, a tek neznatno poboljšanje performansi. Više o tome u [11].

Pregled modela

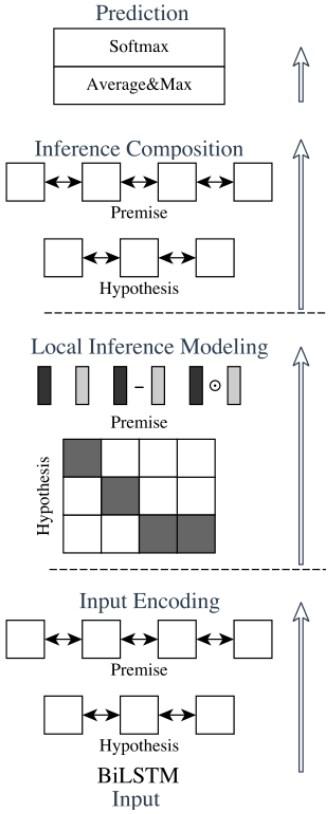
Model se sastoji od tri glavne komponente, koje sekvencijalno obrađuju ulaz: sekvencijalnog enkodiranja ulaza, modeliranja zaključivanja o lokalnoj povezanosti tekstova (pažnja), agregaijcskog/kompozicijskog sloja koji agregira rezultate prethodnih slojeva i klasifikatora (pregled modela po komponentama vidljiv je na slici 3.3). Obzirom na sekvencijalnu prirodu obrade i napredne tehnike enkodiranja konteksta rečenica, ovaj model nazvan je ESIM (eng. *Enhanced Sequential Inference Model*).

Reprezentacija ulaza. Ulaz modelu su rečenice premise i hipoteze reprezentirane unaprijed enkodiranim vektorskim reprezentacijama GloVe ([40]) fiksne duljine, dimenzije 300. Sve vektorske reprezentacije riječi treniraju se zajedno s ostalim parametrima modela.

Sloj za sekvencijalno kodiranje. Prvu obradu predstavlja sloj dvosmjernog LSTM-a, nazvan BiLSTM (eng. *bi-directional LSTM*), koji čita i enkodira premisu i hipotezu istovremeno u smjeru otpočetka prema kraju i obratno. Rezultat svakog vremenskog koraka su, dakle, dva vektora (jedan nastao kretanjem iz jednog, drugi iz drugog smjera). Konkatenacija tih dvaju (skrivenih) vektora predstavlja izlazni vektor odgovarajućeg vremenskog koraka. Umjesto LSTM razmotrene su GRU jedinice ([14]), no LSTM je pokazao bolje performanse.

Neuralna pažnja. Za uspješno zaključivanje o lokalnoj povezanosti, ovaj sloj koristi tzv. meko usklađivanje (eng. *soft alignment*) rečenica. Sličan pristup viđen je u [39] gdje se informacije o svakoj riječi premise dobivaju promatranjem čitave hipoteze po komponentama te sastavljanjem težinskog vektora. Nadogradnjom na taj pristup, ovdje u startu imamo prednost kontekstualnih informacija spremljenih u već enkodiranim reprezentacijama premise i hipoteze od strane BiLSTM-a, za razliku od gotovih enkodiranih vektora koji ne sadrže te kontekstualne informacije. Time dobivamo informacije o ulozi pojedine fraze unutar njene rečenice.

Za meko usklađivanje računamo težine e_{ij} pomoću para vektora skrivenih stanja riječi premise i hipoteze, za sve kombinacije riječi premise i hipoteze. Informacije o lokalnoj povezanosti fraza premise i hipoteze time su zabilježene težinama e_{ij}. Ovo je slično pristupu u poglavlju 3.3, gdje smo informacije o međurečeničnoj povezanosti svake riječi premise računali prolaskom kroz čitavu hipotezu (i obratno). Kako bismo poboljšali zaključivanje o lokalnoj povezanosti, idemo korak dalje.



Slika 3.3: Hijerarhijska organizacija modela. Izvor: [11]

Dodatno učenje lokalnih informacija. Pokazano jest kako enkodirane reprezentacije riječi čuvaju semantičke odnose i nakon primjene aritmetičkih operacija među vektorima ([37]). Konkretno, oduzimanjem bliskih vektora oduzima se njihov kontekst iz izraza. S tom motivacijom, računamo *razliku* te *Hadamardov produkt* (množenje vektora po komponentama) inicijalno enkodiranih vektora i dobivenih težinskih suma. Ovime se nastoji postići veća točnost u prepoznavanju kontradikcije između tekstova (oduzimanje). Rezultate ovih operacija konkateniramo uz postojeće reprezentacije rečenica.

Agregacija i klasifikator. Rezultantni vektor obrađujemo slično kao u prvom koraku, BiLSTM-om koji bilježi sekvencijalne informacije o lokalnom značenju u oba smjera. To se proslijeđuje jednoslojnoj, unaprijednoj neuronskoj mreži s ReLu aktivacijama. Reprezentacije premise i hipoteze se konkateniraju i agregiraju. Umjesto sumacije kao u [39], agregaciju rezultata provodimo računanjem *average-* i *max-poolinga*. Vrši se konkatenacija oba rezultata u konačni vektor fiksne duljine koji se proslijeđuje klasifikatoru.

Klasifikacija se vrši višeslojnom mrežom perceptrona s jednim skrivenim slojem s tangens hiperbolni funkcijom aktivacije i $softmax$ izlaznim slojem. Za funkciju troška koristi se unakrsna entropija.

Hiperparametri i rezultati

Za optimizaciju je korišten Adam optimizator sa standardnim vrijednostima parametara iz rada [34]. Početna brzina učenja je $4 \cdot 10^{-4}$. Svi skriveni slojevi LSTM-a su dimenzije 300. Rezultati i usporedba s drugim relevantnim modelima vidljivi su u tablici 3.3. Modeli su poredati uzlazno po točnosti na testnom skupu SNLI ([10]). Najprije navodimo dva modela od [10], gdje razlikujemo LSTM-baziran model te klasifikator s ručno odabranim

MODEL	TRAIN	TEST	#PARAM.
LSTM [10]	84.4	77.6	221k
Klasifikator [10]	99.7	78.2	-
100D LSTM s pažnjom na razini riječi [41]	85.3	83.5	252k
450D LSTMN s pažnjom [12]	88.5	86.3	3.4M
Razloživi model [39]	89.5	86.3	382k
Razloživi model s unutar-rečeničnom pažnjom [39]	90.5	86.8	582k
600D ESIM [11]	**92.6**	**88.0**	4.3M

Tablica 3.3: Rezultati modela i usporedba s ostalim modelima u ovom radu, kao i drugim relevantnim modelima

značajkama.

Sljedeći po redu jest model s pažnjom [41], koji koristi 100-dimenzionalne LSTM jedinice. Pažnja se u tom modelu koristi prilikom čitanja hipoteze za prolazak nad izlaznim vektorima h_t vremenskih koraka čitanja premise, što omogućuje bolje usklađivanje relevantnih fraza u dvama rečenicama. Varijanta ovog modela s pažnjom na razini riječi, u obliku matrice riječi premise i hipoteze, postiže najbolje performanse.

Ovo je prvo rješenje trenirano s kraja na kraj koje postiže najbolje performanse na RTE zadatku i po performansama prestiže klasifikator baziran na ručno izabranim jezičnim značajkama ([10]). To omogućuje neovisnost modela o jeziku nad kojim je treniran i demonstrira da je kompletno rješenje trenirano s kraja na kraj bazirano na dubokim mrežama kompetitivno za RTE zadatak.

Za referencu je naveden i [12] model koji uz visok broj parametara i korištenje duboke fuzije pažnje postiže najbolje rezultate u vrijeme razloživog modela [39]. Navedeni razloživi model s unutar-rečeničnom pažnjom postiže usporedive rezultate uz znatne uštede u performansama.

ESIM se može smatrati nadogradnjom tog modela jer kompenzira za nedostatak sekvencijalno enkodiranih informacija premise i hipoteze pomoću BiLSTM-a. LSTM u oba modela bilježi rečenični kontekst, a u [41] je pokazano da pažnja nad riječima druge rečenice omogućuje preciznije usklađivanje relevantnih fraza. ESIM ima dodatni sloj kodiranja čitave premise koji je izostao u razloživom modelu. Tako enkodirane riječi bivaju proslijeđene dalje modelu što već u startu daje bolje reprezentacije premise i hipoteze u odnosu na diskretan niz enkodiranih riječi.

Postoji još jedna varijanta ovog modela u [11], koja koristi parsirana sintaktička stabla za rečenične reprezentacije i ima nešto drugačiju arhitekturu koja rezultira poboljšanjem u odnosu na ESIM, točnost od **88.6%**. Ovo poboljšanje dolazi po cijenu korištenja mnogo kompleksnije arhutekture, ali i po cijenu potrebe korištenja gotovog sintaksnog parsera koji je ovisan o jeziku skupa podataka.

Poglavlje 4

Eksperimenti

U ovom poglavlju opisujemo eksperimente provedene implementacijom ESIM modela ([11]) opisanog u poglavlju 3.4, u Kerasu. Najprije spominjemo promjene koje je bilo potrebno napraviti da bi korištena Keras implementacija odgovarala onoj u [11]. Takvu implementaciju testiramo nad SNLI kako bismo postavili referentni rezultat. Potom vršimo Bayesovu optimizaciju hiperparametara pretražujući prostor hiperparametara s ciljem poboljšanja performansi modela i opisujemo postignute rezultate. Naposljetku, najboljim modelom iz prethodnog koraka testiramo prijenos znanja provjeravajući njegov uspjeh nad kombinacijom MNLI i SNLI skupova te analiziramo rezultate i uspoređujemo ih s modelom iz MNLI rada ([52]).

4.1 Implementacija modela

Originalno, sekvencijalni model ESIM iz [11] implementiran je u Theanu. *Theano* ([51]) je Python biblioteka za efikasno izračunavanje matematičkih izraza s podrškom za brze izračune pri radu s tenzorima. Budući da je Theano općeniti okvir za numeričko računanje, dok Keras predstavlja sučelje više razine za objektno orijentiran rad s modelima dubokog učenja kako je opisano u poglavlju 2.5 te uzimajući u obzir da je u vrijeme pisanja ovog rada (2018.) najavljen prestanak razvoja Theana[1], za implementaciju ESIM modela korištena je implementacija u Kerasu preuzeta iz [55].

Korištena implementacija zahtijevala je određene dorade jer model, nakon preuzimanja i testiranja, postiže točnost od 82.8% na testnom skupu, što je mnogo niže od točnosti originalnog rada (88% [11]). Nakon usporedbe s izvornim kodom u Theanu iz [11] utvrđeno je da polazna Keras implementacija ([55]) ima različitu arhitekturu: drugačiji razmještaj postojećih slojeva, jedan sloj za ispuštanje više nego izvorni Theano kod i koristi različite vrijednosti hiperparametara (veličina serije, optimizacijska funkcija, stopa učenja, stopa

[1] `https://groups.google.com/forum/#!msg/theano-users/7Poq8BZutbY/rNCIfvAEAwAJ`

ispuštanja, *L2* regularizacijska stopa, stopa obrezivanja gradijenta, inicijalizacija stanja LSTM-a, treniranje vektorskih projekcija itd.). Pored toga, polazni model ima različito napravljeno predprocesiranje podataka od onog opisanog u radu te ne trenira vektorske reprezentacije riječi, kako je navedeno u radu [11].

Usklađivanjem navedenih komponenti, izmjenom arhitekture te promjenom hiperparametara modela tako da odgovaraju onima u originalnom ESIM modelu ([11]), Keras ESIM model postiže točnost od 86.64% na validacijskom skupu. Ovaj model koristimo dalje za optimizaciju hiperparametara.

4.2 Bayesova optimizacija hiperparametara

Tehnički detalji optimizacije

Za optimizaciju hiperparametara korištena je *Hyperopt* ([8]), biblioteka u Pythonu za Bayesovu optimizaciju hiperparametara koja koristi TPE vjerojatnosni model spomenut u poglavlju 2.4. Prostor hiperparametara opisan je u tablici 4.1.

Hiperparametri s diskretnim vrijednostima birani su iz skupa izabranih vrijednosti za taj parametar oblika $\{x_1, x_2, \ldots, x_n\}$. Samo jedan hiperparametar biran je iz intervala realnih brojeva oblika $[a, b]$, odakle su vrijednosti birane nasumično uniformnom distribucijom.

U posljednjem sloju ispuštanja isprobana je umanjena vrijednost ispuštanja (faktor 0.25) u odnosu na stopu ispuštanja u svim ostalim slojevima (0.5) jer je takva modifikacija uočena u Keras implementaciji ESIM-a iz [55]. Taj hiperparametar naveden je kao *"Posljednji faktor ispuštanja"* u tablici.

HIPERPARAMETAR	MOGUĆE VRIJEDNOSTI
Veličina serije	$\{128, 512\}$
Posljednji faktor ispuštanja	$\{0.5, 0.25\}$
L2 regularizacija	$\left\{0.0, 10^{-5}\right\}$
Početna stopa učenja	$\left\{10^{-4}, 4 \cdot 10^{-4}, 10^{-3}\right\}$
Optimizacijska funkcija	{RMSprop, Adam, Nadam}
Kraćenje gradijenta	uniformna distribucija nad $[0.0, 15.0]$

Tablica 4.1: Prostor hiperparametara koji pretražujemo

Kao optimizacijski algoritmi učenja isprobane su varijante RMSprop ([42]), Adam ([34]) i Nadam (Adam optimizator s Nesterov ([49]) modifikacijom zamaha ([24])). Motivacija za odabir baš ovih algoritama jest što RMSprop rješava pojavu nestajućih stopa učenja Adagrad algoritma ([42]). Adam osim toga ima i parametar eksponencijalnog ras-

pada prethodnih gradijenata, a uz to je preporučen parametar u originalnom modelu [11]. Nadam pruža alternativu koristeći Nesterov zamah na Adam algoritmu.

Za veličinu serije isprobane su dvije vrijednosti (128 i 512). Budući da smanjenje veličine serije za faktor 2 u prosjeku udvostručuje vrijeme treniranja, izabrane su dvije vrijednosti tog hiperparametra dovoljno različite da zabilježe moguću razliku u performansama, a dovoljno velike da treniranje bude vremenski isplativo. *L*2 regularizacija dodana je kao opcionalna tehnika regularizacije, uz ispuštanje, s mogućim vrijednostima 10^{-5} (vrijednost iz Keras implementacije [55]) i 0 (nema *L*2 regularizacije), kao u originalnom Theano kodu. Početne stope učenja su preporučena stopa iz originalnog rada ($4 \cdot 10^{-4}$), a alternative su uobičajene vrijednosti tog hiperparametra: 10^{-4} i 10^{-3}. Originalni kod sadrži hiperparametar kraćenja gradijenta iznosa 10. Stoga je za raspon vrijednosti uzet interval realnih brojeva oko te vrijednosti (koji uključuje 0, što znači da nema kraćenja).

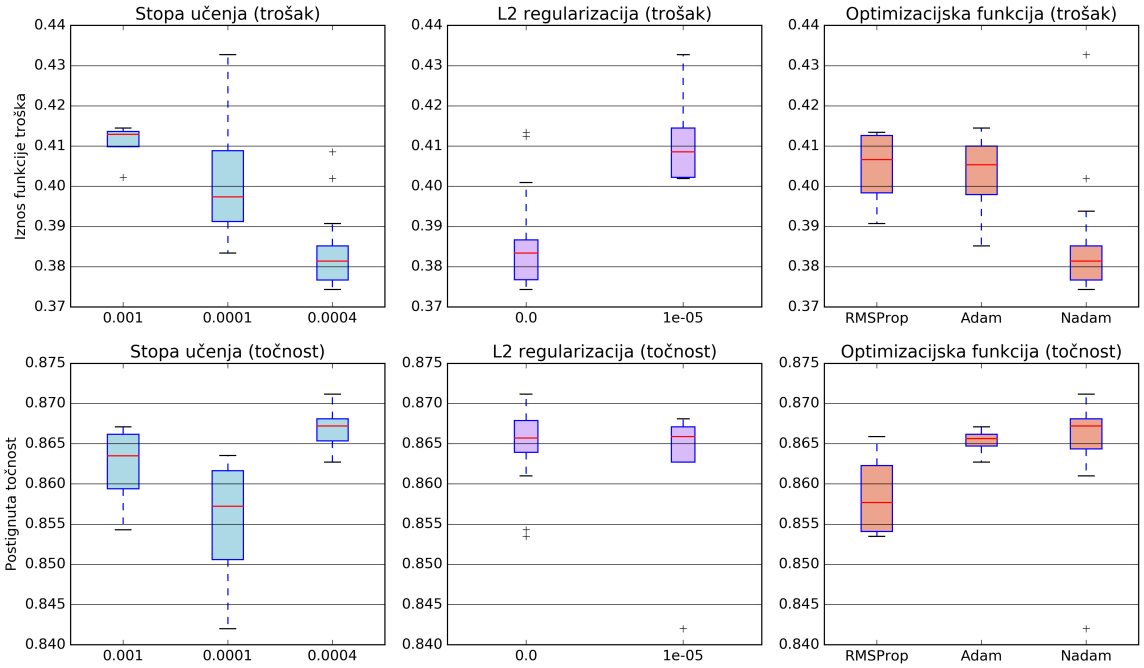
Treniranje modela vođeno je stohastičkim gradijentnim spustom kojim minimiziramo funkciju troška (višeklasna unakrsna entropija kao u [11]). Na kraju epohe računamo vrijednost funkcije troška na validacijskom skupu. Učenje se usporava za red veličine (faktor 10) ako 5 uzastopnih epoha vrijednost funkcije troška nije smanjena. Nakon 7 takvih uzastopnih epoha, učenje se zaustavlja (kriterij ranijeg zaustavljanja preuzet iz Theano koda [11]).

Tijekom Bayesove optimizacije, surogat funkcija koju minimiziramo ovisi o vrijednosti funkcije troška na validacijskom skupu. Na ovaj način, pri Bayesovoj optimizaciji ne biramo model s najvećom točnošću nego onaj s najmanjom vrijednošću funkcije troška. Cilj ovakvog pristupa jest izabrati model koji manje griješi jer je pretpostavka da će se to prevesti u bolju generalizaciju na neviđenom (testnom) skupu.

Samo uzimajući u obzir diskretne hiperparametre, broj mogućih konfiguracija je 72. Bayesovu optimizaciju pokrećemo kroz 25 iteracija, a konfiguracije u svakom koraku bira TPE vjerojatnosni model Hyperopt-a.

Rezultati Bayesove optimizacije

Distribuciju vrijednosti hiperparametara vidimo na vizualizacijama 4.1, 4.2 i 4.3. Vizualizacije diskretnih parametara napravljene su u obliku dijagramaa s pravokutnikom iz kojih se daju iščitati podaci o distribuciji vrijednosti kroz eksperimente. Medijan je prikazan kao vodoravna crta na sredini pravokutnika. Pravokutnik obuhvaća vrijednosti između donjeg i gornjeg kvartila, a vodoravne crte iznad te ispod pravokutnika povezane isprekidanom okomitom linijom s pravokutnikom predstavljaju najveću i najmanju vrijednost koje se nalaze unutar 1.5 × (interkvartilnog raspona). Vrijednosti izvan tog raspona zabilježene su posebno simbolima + te se smatraju vrijednostima koje odudaraju od ostalih. Na svakom dijagramu prikazane su distribucije hiperparametara obzirom na funkciju troška (gornji red, manje je bolje) i točnost (donji red, više je bolje) na validacijskom skupu.



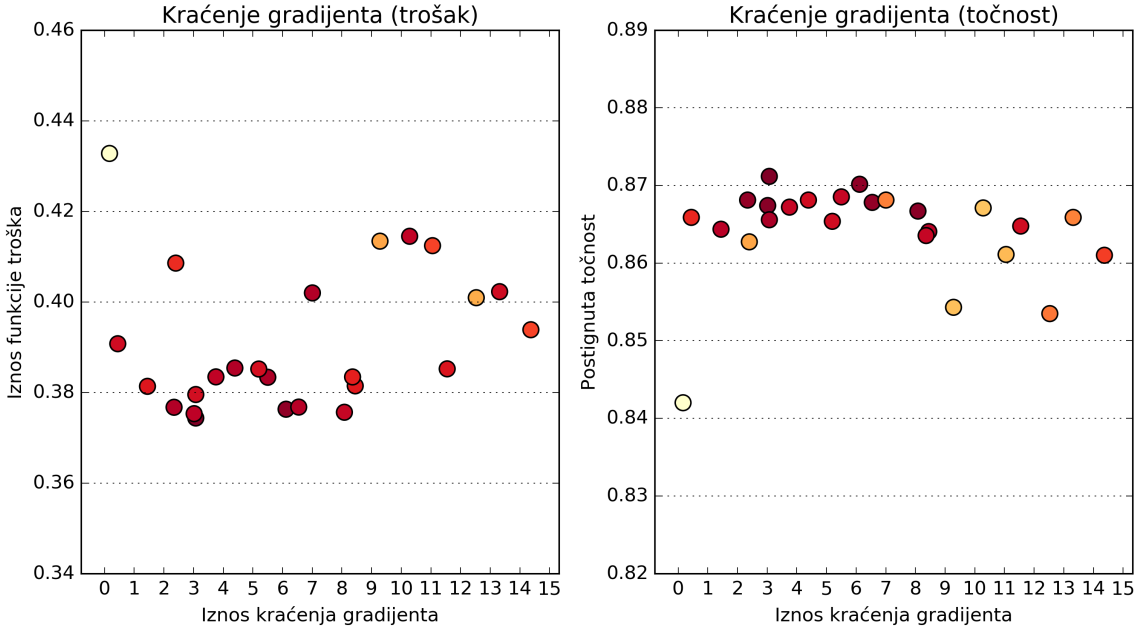
Slika 4.1: Distribucija hiperparametara u prostoru hiperparametara. Vizualizacije izrađene korištenjem [31].

Na slici 4.1 nalaze se hiperparametri koji su više utjecali na uspjeh modela. Kod funkcije troška vidljiva je velika razlika između različitih stopa učenja. Niža stopa učenja sporije konvergira pa lakše nalazi lokalne minimume funkcije troška. Međutim, postoji balans pa tako najuspješnija nije najmanja stopa učenja (10^{-4}) nego ona iz originalnog rada ($4 \cdot 10^{-4}$), koja postiže uvjerljivo najmanje vrijednosti funkcije troška. Izabrana stopa postiže i najveće točnosti te je većina vrijednosti grupirana iznad pravokutnika većine vrijednosti druge najuspješnije stope.

Kod $L2$ regularizacije, razlika u točnosti je minimalna, ali razlika u minimizaciji funkcije troška je znatna. Pravokutnici distribucije vrijednosti ovog parametra udaljeni su jedan od drugoga, što ukazuje na znatnu razliku između modela s različitim vrijednostima ovog parametra. Donje vrijednosti funkcije troška koje model postiže za vrijednost 10^{-5} su blizu maksimuma vrijednosti za parametar 0, što ukazuje na znatnu razliku u performansama u korist ne korištenja $L2$ regularizacije. Smanjen utjecaj $L2$ regularizacije moguće je objasniti postojećom regularizacijom u vidu ispuštanja. Moguće je da dodatna $L2$ regularizacija previše ograničava model.

Što se tiče funkcije optimizacije učenja, mjera točnosti je opet zavaravajuća. Najprije, velika je razlika između RMSprop i druga dva algoritma, tolika da samo najveće vrijednosti RMSprop-a dostižu prosječne vrijednosti druga dva algoritma, koji su blizu u većini slučа-

jeva, ali Nadam ima veća odskakanja od medijana. Prava razlika vidi se tek kod funkcije troška, gdje RMSprop i Adam postižu podjednake rezultate, dok je većina vrijednosti kod Nadam algoritma ispod minimuma obiju drugih distribucija, što znači da modeli s Nadam algoritmom optimizacije konzistentno postižu nižu grešku. Kako optimizacijska funkcija usmjerava stopu učenja i kretanje u smjeru gradijenta, ima smisla da ona ima velik učinak na performanse nad funkcijom troška.

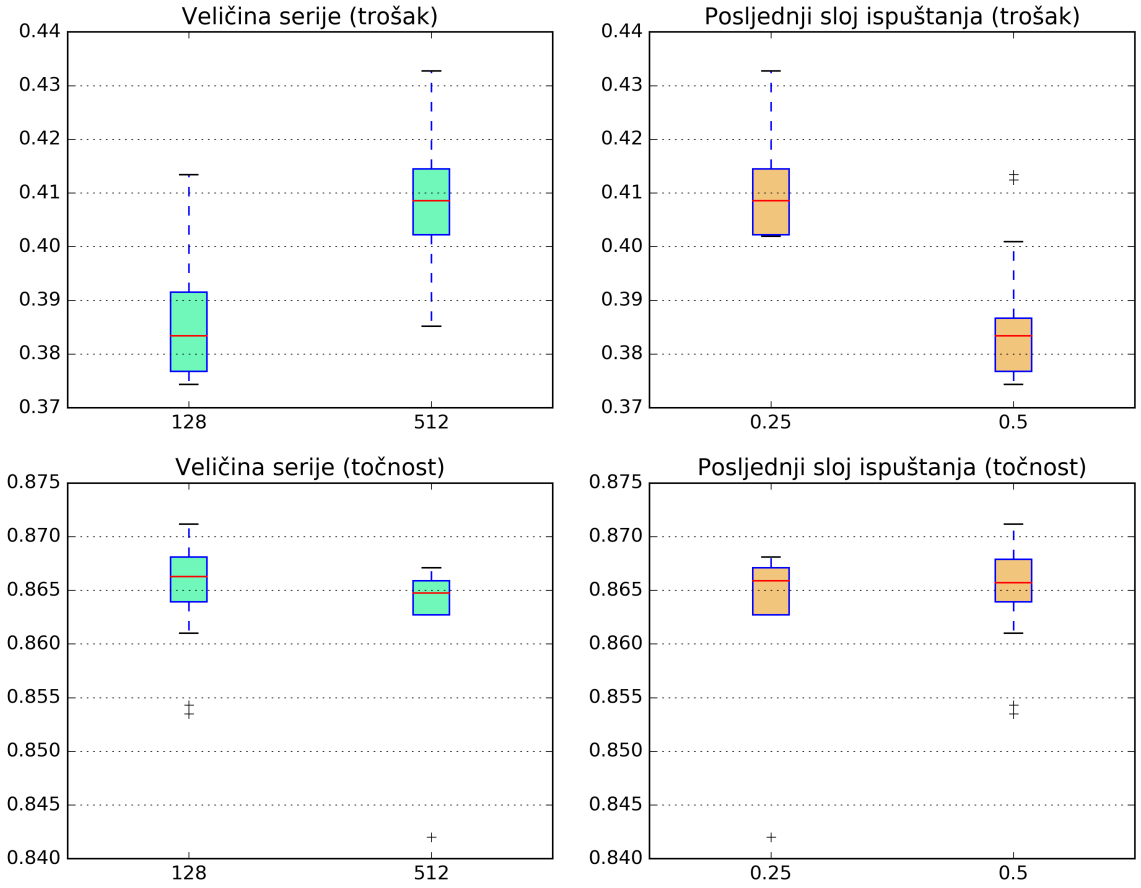


Slika 4.2: Distribucija vrijednosti kraćenja gradijenta u intervalu [0, 15]. Lijevo je prikazan odnos iznosa kraćenja i funkcije troška, a desno odnos iznosa kraćenja i točnosti modela. Na lijevom grafu tamnija boja predstavlja *višu točnost*, a na desnom tamnija boja *nižu grešku* (u oba slučaja, tamnije je bolje). Vizualizacije izrađene korištenjem [31].

Sljedeće promatramo distribuciju koeficijenta kraćenja gradijenta na slici 4.2. Za funkciju troška, najbolje vrijednosti su grupirane između 2 i 9, sa zanimljivom krivuljom kretanja uspješnih vrijednosti. Optimalnima se pokazuju vrijednosti grupirane oko 3 i 6, s vrijednostima prije i poslije toga koje odskaču prema gore. Kako se udaljavamo od tog intervala, greška znatno raste (primjer vrijednosti blizu 0 i oko 10). Kod točnosti, razlike nisu toliko očite, no opet odskaču vrijednosti 3 i 6, s tim da je ovdje vrijednost oko 3 vidljivo uspješnija. Ponovno su najlošije vrijednosti oko 0 i 10, što nam daje prostora za poboljšanje nad originalnom implementacijom jer je tamo preporučena vrijednost upravo 10.0.

Uočljivo je kako je oko optimalnih vrijednosti veći broj točaka, dok su suboptimalne

vrijednosti raspršene posvuda - to je upravo Bayesova optimizacija na djelu, koja ne ispituje prostor hiperparametara nasumično, nego bira sljedeće vrijednosti hiperparametara na temelju prethodno poznatih uspješnih vrijednosti.



Slika 4.3: Distribucija hiperparametara u prostoru hiperparametara. Vizualizacije izrađene korištenjem [31].

Promotrimo i raspodjelu posljednja dva hiperparametra na slici 4.3. Velik raspon vrijednosti obje vrijednosti veličine serije (gornji red) ukazuje da uspjeh modela više ovisi o drugim hiperparametrima. Ipak, vrijednosti za manju veličinu serije (128) grupirane su oko nešto niže vrijednosti troška. Kod točnosti je razlika minimalna, kao i u slučaju stope ispuštanja.

Promatrajući distribuciju vrijednosti stope ispuštanja posljednjeg sloja obzirom na trošak, dalo bi se zaključiti da niža stopa (0.25) utječe negativno na uspjeh (pravokutnik smješten znatno više na grafu troška). Međutim, u praksi je u svakom od 5 eksperimenata gdje se pojavila stopa ispuštanja posljednjeg sloja 0.25, ta vrijednost bila uparena s vrijednošću

hiperparametra *L2* regularizacije od 10^{-5}. Ta vrijednost *L2* regularizacije pojavljuje se i zasebno te pokazuje negativan učinak na performanse modela (kako je vidljivo na slici 4.1). Međutim, o posljednjoj stopi ispuštanja ne možemo to zaključiti, nego samo da ta dva parametra u kombinaciji negativno utječu na performanse modela jer za zaključivanje o sâmoj stopi ispuštanja 0.25 nemamo dovoljno izoliranih eksperimenata. Kao i kod veličine serije, i ovdje vidimo veliku raspršenost vrijednosti, koja ukazuje da točnost više ovisi o drugim hiperparametrima nego o ovome.

HIPERPARAMETAR	POLAZNI MODEL	BAYES OPT. MODEL
Veličina serije	32	128
Posljednji faktor ispuštanja	0.5	0.5
L2 regularizacija	0.0	0.0
Početna stopa učenja	$4 \cdot 10^{-4}$	$4 \cdot 10^{-4}$
Optimizacijska funkcija	Adam	Nadam
Kraćenje gradijenta	10.0	3.07
Najmanja greška (val)	0.382	**0.374**
Najveća točnost (val)	86.64%	**87.11%**
Točnost (test)	85.40%	**85.91%**

Tablica 4.2: Usporedba performansi modela s originalnim i optimiziranim hiperparametrima.

Dio hiperparametara iz rada je održan (početna stopa učenja, faktor ispuštanja u posljednjem sloju, stopa *L2* regularizacije). Međutim, Bayesova optimizacija je pokazala poboljšanje kod odabira različitih vrijednosti za **kraćenje gradijenta** i **optimizacijski algoritam**. Usporedbu najboljeg i početnog modela vidimo u tablici 4.2.

Razlike u performansama nisu goleme, ali su uočljive. Priložene su stope točnosti nad validacijskim i testnim skupom (testiranje nad testnim skupom oba modela provedeno nakon Bayes optimizacije). Priložene su i vrijednosti funkcije troška. Obzirom da su glavne razlike između modela u kraćenju gradijenta i optimizacijskoj funkciji, najveća je razlika upravo na performansama funkcije troška, što se indirektno prevodi i u veću točnost nad validacijskim odnosno testnim skupom.

Pošto bolje rezultate postižu konfiguracije s manjom veličinom serije (128), najuspješniji model treniran je sa istom konfiguracijom parametara i veličinom serije 32. Točnost na validacijskom skupu takvog modela podjednaka je modelu s veličinom serije 128, s nešto višom vrijednošću funkcije troška, pa je za daljnje testiranje korišten model s veličinom serije 128 iz tablice 4.2 (desno).

Naposljetku, usporedimo dobivenu točnost Keras implementacije s onima iz originalnog rada [11]. Izvorni kod tog rada pisan je u Theanu, dok je ovdje korišten model napisan u Kerasu (s TensorFlowom u pozadini). Implementacija ESIM modela korištena u [52]

pisana je u TensorFlowu, a postiže **86.7%**, što je 1.3% niže u odnosu na originalnu Theano implementaciju (88.0%). Theano i TensorFlow omogućuju vrlo preciznu razinu kontrole nad implementacijom, dok Keras kao apstrakcijski sloj (s TensorFlowom u pozadini) nudi gotove gradivne elemente za neuronske mreže pa je moguće da je zbog toga na Keras implementaciji izgubljeno na performansama (postignutih **85.91%**, razlika od 0.79% u odnosu na TensorFlow implementaciju). Također, pošto detalji preprocesiranja podataka i arhitekture slojeva nisu detaljno navedeni u radu [11] nego su tumačeni direktno iz izvornog koda, tu su moguće minimalne razlike. Naposljetku, pri Bayesovoj optimizaciji hiperparametara i treniranju modela, kriterij uspjeha i ranijeg zaustavljanja bile su performanse funkcije troška na validacijskom skupu jer je motivacija toga bila bolja generalizacija na neviđeni, testni skup. Moguće je da bi se korištenjem točnosti kao glavnog kriterija postigli nešto viši rezultati točnosti.

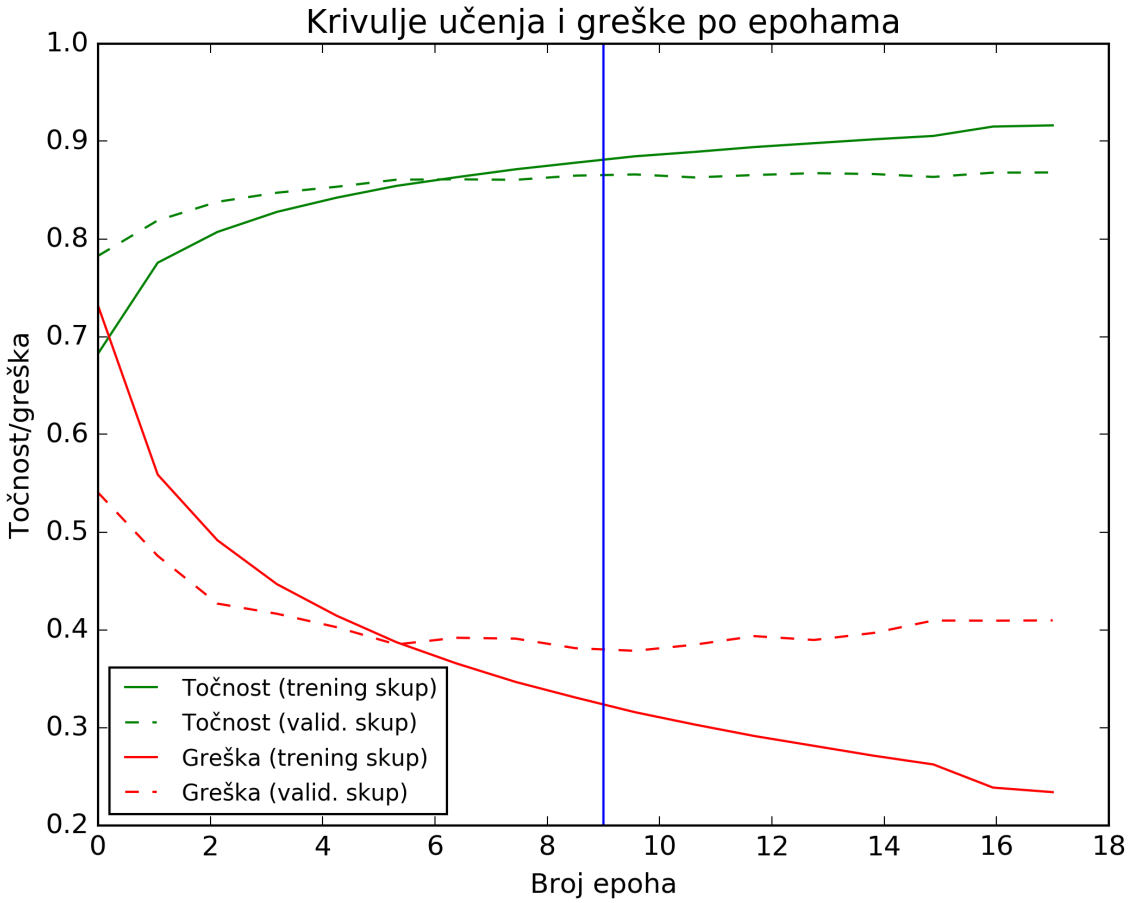
Unaprijeđenje rezultata

Prednost Bayesove optimizacije jest što pruža učinkovit način isprobavanja većeg broja različitih vrijednosti hiperparametara i njihovih kombinacija na uspjeh modela. No, teško je zaključiti samo iz tih podataka o direktnom učinku vrijednosti pojedinih hiperparametara na uspjeh modela obzirom da je u svakom koraku birana čitava konfiguracija različitih vrijednosti hiperparametara. Kako bi se moglo više zaključiti o direktnom utjecaju pojedinog hiperparametra na uspjeh modela, potrebno je napraviti eksperimente s izoliranim promjenama vrijednosti pojedinih hiperparametara uz fiksne vrijednosti ostalih. Ovo može istaknuti hiperparametre koji imaju veći utjecaj, što se da iskoristiti za daljnju optimizaciju vrijednosti upravo tih hiperparametara.

Bayesova optimizacija može se nastaviti s ovim rezultatima. Hiperparametri koji utječu više mogu se nastaviti optimizirati s novim rasponom vrijednosti (ispuštanje, stopa učenja, optimizacijska funkcija). Također, mogu se uvesti novi, neisprobani parametri: usporavanje učenja, drugačiji kriteriji ranijeg zaustavljanja, promjena hiperparametara optimizacijske funkcije. Moguće je i promijeniti parametre Bayesove optimizacije: smanjiti period treniranja pojedinog modela kako bi se mogao isprobati veći broj konfiguracija. Naposljetku, potrebno je napraviti statističke testove rezultata kako bi se preciznije utvrdio učinak optimizacije na dobivene rezultate.

4.3 Prijenos znanja

Prijenos znanja modela provjeravamo nizom testova nad dva skupa podataka: SNLI i MNLI. Cilj je provjeriti kakve performanse ostvaruje model prilagođen jednom testnom skupu nad drugim skupovima. Najprije testiramo točnosti najboljeg modela naučenog na SNLI-u, nad MNLI skupom. Potom, isti model treniramo nad MNLI skupom i testiramo



Slika 4.4: **Krivulja učenja** najboljeg modela na temelju Bayes optimizacije (tablica 4.2 desno) koja prikazuje kretanje točnosti i greške nad trening i validacijskim skupovima, evaluirano na kraju svake epohe. Nakon dosezanja platoa funkcije troška, učenje se zaustavlja, a koriste se parametri iz najbolje epohe obzirom na funkciju troška. Vidimo da se dosezanje donjeg platoa funkcije troška otprilike poklapa sa dosezanjem platoa obzirom na točnost na validacijskom skupu. Vrijednosti nakon toga za trening i validacijski skup se znatno razilaze, što ukazuje na moguću pretreniranost u kasnijim epohama na trening skup. Vizualizacije izrađene korištenjem [31]

ga nad dvije varijante MNLI validacijskog skupa: usklađenom (iste kategorije kao trening skup) i neusklađenom (neviđene kategorije) te SNLI-em. Naposljetku treniramo model nad združenim MNLI i SNLI skupom te provjeravamo uspjeh nad svim skupovima te podkategorijama MNLI skupa. Kao referentni rezultat navodimo performanse ESIM modela iz [52] jer su u tom radu rađeni slični eksperimenti nad kombinacijama ovih dvaju skupova.

Metodologija treniranja

Treniranje se vrši prateći performanse greške na validacijskom skupu, a učenje se zaustavlja kada 7 epoha zaredom nema poboljšanja nad funkcijom troška. Testiranje nad MNLI skupom vrši se nad validacijskim skupovima (usklađeni i neusklađeni) jer testni skup nije javno dostupan. Zbog toga pri odabiru najboljeg modela uzimamo u obzir samo funkciju troška te biramo onaj s minimalnom funkcijom troška tijekom svih epoha treniranja (slika 4.4).

Na slici vidimo da se performanse na trening i validacijskom skupu tek blago razilaze za izabranu epohu te da je točka najmanje generalizacijske greške gotovo dosegnuta par epoha ranije. Međutim, više se isplati uzeti model kasnije epohe nego onaj prve epohe gdje se poklapaju funkcije troška nad ta dva skupa jer se kasnije obično nađe još niži minimum (epoha 9 na slici 4.4) pa se može postići dodatna točnost bez znatnog gubitka na generalizaciji (vidljivo je da krivulje nakon dosezanja minimuma troška više manje stagniraju za validacijski skup). Bez kriterija ranijeg zaustavljanja, nakon nekoliko epoha stagnacije, greška na validacijskom skupu ponovno počinje rasti (nije vidljivo na grafu jer je ovdje primijenjen kriterij ranijeg zaustavljanja).

Tako izabranom modelu mjerimo točnost nad validacijskim skupom. Testiranje nad SNLI-em vršimo na testnom skupu. Rezultate modela vidimo u tablici 4.3.

Učenje nad jednim skupom

ESIM model treniran nad SNLI, a evaluiran nad MNLI pokazuje mnogo niže performanse nego što postiže nad SNLI skupom (prvi stupac) što se pripisuje različitosti tih dvaju skupova. Leksička raznolikost MNLI skupa predstavlja prevelik izazov modelu koji nije učen nad takvim tipom rečenica - isprekidane, nepotpune (razgovorni jezik), s drugačijim glagolskim vremenima i duljim rečenicama. Tijekom treniranja nad SNLI skupom, broj riječi u vokabularu za koje postoje GloVe reprezentacije bio je 56K, dok je pri treniranju nad MNLI-em veličina vokabulara bila oko 175K (trostruko više).

Duljina rečenica uzeta je u obzir zbog različitosti skupova. Originalni kod [11] sadrži najveću dozvoljenu duljinu rečenice 42 jer su sve rečenice kraće od toga (prosječna duljina premise u SNLI je 14.1 riječi, a hipoteze 8.3). U MNLI-u prosjek premise je 22.3 uz najdulju premisu od 401 riječi, dok su hipoteze nešto kraće s prosjekom od 11.4 (najdulja 70 riječi). Zbog ovih razlika, modeli trenirani nad SNLI-em koriste originalnu vrijednost 42 za duljinu rečenice, dok za treniranje nad MNLI koristimo vrijednost 50, što je preporučena vrijednost autora tog skupa [52]. Rečenice dulje od toga skaćene su na tu duljinu. Autori također tvrde da povećanje duljine hipoteze ne utječe pozitivno na rezultate, dok povećanje duljine premise utječe blago negativno. Stoga je isti model treniran i sa znatno većom duljinom rečenica 150, uz rezultate u zagradi pored istih rezultata za duljinu od 50. Jedino nad neusklađenim skupom vidi se neznatan napredak (0.3%), dok je drugdje vidljiv

	Keras ESIM treniran nad			ESIM iz [52]
Evaluacijski skup	SNLI	MNLI	MNLI+SNLI	MNLI+SNLI
SNLI	**85.91%**	58.32% (56.51%)	78.19%	79.7%
MNLI (usklađeni)	57.42%	**71.51%** (70.85%)	71.32%	72.4%
MNLI (neusklađeni)	58.24%	70.93% (71.23%)	**71.73%**	71.9%

Tablica 4.3: **Rezultati različitih kombinacija** učenja i testiranja nad MNLI i SNLI skupovima. **Istaknuti** su najbolji rezultati za svaki trening skup Keras modela. U zagradi su navedeni rezultati za duljinu rečenica do 150 riječi. Za referencu navodimo rezultate implementacije ESIM modela iz MNLI rada ([52]).

pad performansi, posebno nad SNLI. Stoga za ostale eksperimente provodimo testove s najvećom duljinom 50.

Sljedeće promatramo **treniranje nad MNLI** te testiranje nad SNLI i varijantama MNLI validacijskog skupa. Razlika između usklađenog i neusklađenog skupa nije previše izražena. Očito postojeće kategorije u trening skupu dobro modeliraju jezičnu raznolikost neusklađenog validacijskog skupa. Zanimljivo je promotriti performanse nad SNLI skupom. To je varijanta neusklađenog testiranja jer MNLI ne sadrži kategoriju tekstova koja bi odgovarala SNLI jezičnoj strukturi.

Ispostavlja se da je teže zaključivati o SNLI bez prethodnog učenja na tom tipu rečenica nego o drugim neusklađenim kategorijama jer model treniran nad MNLI-em postiže nad SNLI-em znatno nižu točnost u odnosu na MNLI neusklađeni skup, premda oba skupa sadrže tipove rečenica nad kakvima model nije učio. Ovo je objašnjivo kategorijama u MNLI neusklađenom skupu, koje su strukturno slične nekim kategorijama u usklađenom skupu, npr. transkripti razgovora licem u lice (*"Face-to-face"*) te vladin izvještaj o terorističkim napadima (*"9/11"*) iz neusklađenog skupa slični su transkriptima telefonskih razgovora (*"Telephone"*), odnosno dokumentima s vladinih službenih stranica (*"Government"*), tako da model treniran nad MNLI-em dobro prenosi naučeno na neusklađeni skup.

Uočljivo je također kako model treniran nad SNLI, postiže nad MNLI kategorijama (koje predstavljaju tip neusklađenog evaluacijskog skupa) otprilike iste performanse kao model treniran nad MNLI, a evaluiran nad SNLI (također neviđen tip rečenica) - otprilike 58% u oba slučaja.

Učenje nad združenim skupom

Naposljetku, treniramo model nad združenim SNLI i MNLI skupom. Pošto je SNLI strukturno različit od MNLI-a tako što je MNLI organiziran u tematske kategorije, a čitav SNLI je tematski jedna kategorija (kako je objašnjeno u poglavlju 1.4), SNLI smatramo dodatnom kategorijom MNLI-a za potrebe treniranja. Obzirom da je SNLI neproporcionalno

veći od bilo koje druge kategorije u MNLI (549K u odnosu na 77K-83K primjera), pri treniranju koristimo podskup ukupnog broja primjera kao u [52]. Za eksperiment probiremo **nasumičnih 15% primjera** (oko 82K) iz trening skupa SNLI-a te ih dodajemo kao novu kategoriju u trening skup MNLI-a. Ovo osigurava da model nema preveliku sklonost prema SNLI tipu rečenica jer je svaka kategorija podjednako zastupljena. Rezultantni skup sadrži 475K rečeničnih parova.

Pri treniranju nad MNLI-em, usmjeravanje učenja (kriterij usporavanja i ranijeg zaustavljanja učenja) bilo je vršeno na kraju svake epohe provjerom vrijednosti funkcije troška nad validacijskim skupom (usklađenim - koji sadrži kategorije iz trening skupa). Obzirom da je u trening skup uključena nova kategorija (SNLI), za usmjeravanje združenog učenja u MNLI validacijski skup dodajemo **nasumično izabranih 2K primjera** iz SNLI validacijskog skupa. Rezultantni MNLI validacijski skup za usmjeravanje učenja ima po 2K primjera iz svake kategorije (njih ukupno 6 sa SNLI kategorijom).

Kako bismo dobili pouzdanije rezultate zbog izbora nasumičnih primjera, eksperiment provodimo 10 puta, svaki put nasumično birajući primjere iz SNLI trening te validacijskog skupa. Rezultati su vidljivi usporedno s rezultatima modela nad pojedinačnim skupovima, u tablici 4.3.

U odnosu na treniranje na MNLI skupom, združeni skup MNLI i SNLI ne donosi znatna poboljšanja na MNLI skupu. Na usklađenim kategorijama vidljiv je čak blagi pad performansi (0.19% razlike), dok nad neusklađenim kategorijama vidimo veći rast (0.8%). Najveću razliku vidimo nad SNLI skupom, skok od gotovo 20%, što se da objasniti mogućom sklonošću modela prema tom skupu, obzirom da je model konstruiran upravo mjerenjem performansi nad tim skupom.

Usporedimo još rezultate združenog optimiziranog Keras modela sa združeno treniranim ESIM modelom iz [52]. Rezultati Keras ESIM modela su ispod rezultata ESIM-a iz [52], što je u skladu s očekivanjima jer Keras implementacija ESIM modela postiže <1% nižu točnost nad SNLI-em (85.91%) u odnosu [52] (86.7%). Združivanje skupova za treniranje prati blagi porast performansi tog modela, zadržavajući zaostatak u prosjeku <1% kroz različite skupove, s najvećom razlikom od 1.5% nad SNLI skupom (prvi red), a najmanjom za neusklađeni MNLI skup (0.17%).

Rezultati po pojedinačnim kategorijama

Prijenos znanja provjeren je i nad individualnim kategorijama MNLI-a. Model treniran nad pojedinačnim skupovima SNLI, MNLI kao i onaj treniran nad združenim skupom ta dva skupa testirani su na svih 10 kategorija MNLI-a te nad SNLI-em. Zbog toga što je podskup SNLI-a dodan u združeni trening skup, SNLI testni skup smatran je *usklađenom* kategorijom skupa za testiranje. Ovo su rezultatni već spomenutog združenog modela koji je treniran i evaluiran 10 puta, tako da je u tablici (za združeni model) naveden prosječan

	Keras ESIM treniran nad			ESIM iz [52]
Evaluacijski skup	SNLI	MNLI	MNLI+SNLI	MNLI
Fiction	57.17%	69.18%	**69.77%**	73.0%
Government	59.79%	77.07%	76.33%	74.8%
Slate	54.68%	65.68%	**66.95%**	67.9%
Telephone	57.22%	72.83%	71.73%	72.2%
Travel	58.25%	72.82%	71.83%	73.7%
SNLI	85.91%	58.32%	**78.19%**	60.7%
MNLI (usklađeni)	57.42%	71.51%	71.32%	72.3%
9/11	57.80%	70.41%	**70.80%**	71.9%
Face-to-face	58.05%	70.67%	**72.16%**	71.2%
Letters	63.02%	74.35%	**74.65%**	74.7%
OUP	57.47%	71.20%	**71.38%**	71.7%
Verbatim	54.78%	67.98%	**69.61%**	71.9%
MNLI (neusklađeni)	58.24%	70.93%	**71.73%**	72.1%

Tablica 4.4: **Performanse po kategorijama** te nad čitavim validacijskim (MNLI) odnosno testnim (SNLI) skupovima za modele trenirane po pojedinačnim, odnosno združenim skupom. **Istaknuti** su rezultati na kojima združeni model pokazuje bolje performanse nego model nad pojedinačnim skupom, a podcrtani su rezultati kod kojih je Keras ESIM bolji nego ESIM iz [52] nad istim skupom.

rezultat 10 eksperimenata za svaku kategoriju.

Performanse po kategorijama navedeni su u tablici 4.4. Referentni ESIM model iz [52] treniran nad združenim skupom nije testiran nad pojedinačnim kategorijama u tom radu, pa u zablici 4.4 po kategorijama navodimo rezultate modela iz tog rada treniranog nad samo MNLI skupom, za koji imamo rezultate po kategorijama. Nazivi kategorija su izvorni engleski nazivi, a svaka od njih je pojašnjena u poglavlju 1.4.

Rezultati združenog modela nad SNLI-em pokazuju znatno poboljšanje u odnosu na samo treniranje nad MNLI-em, ali ne kao treniranje nad samo SNLI-em. Očito, premda MNLI odlikuje velika raznolikost tekstova, oni još uvijek ne modeliraju dovoljno dobro jezičnu strukturu SNLI-a, pa SNLI predstavlja dobar dodatak MNLI skupu kao dodatna kategorija.

Nad usklađenim validacijskim skupom MNLI-a postoji blagi pad točnosti u prosjeku (0.2%), s poboljšanjem nad samo dvije od pet kategorija (*"Fiction"* i *"Slate"*). Pad performansi uočljiv je kod *"Government"*, *"Telephone"* i *"Travel"* kategorija, u prosjeku <1%.

S druge strane, *nad svim neusklađenim kategorijama* model pokazuje bolje performanse. Rast je u prosjeku 0.8%, s najvećim rastom performansi nad *"Face-to-face"* i *"Verbatim"* kategorijama, po 1.6%. Rezultat upućuje na zaključak da dodavanje SNLI

skupa MNLI-u kao dodatne kategorije, ponajprije unaprijeđuje zaključivanje modela nad neusklađenim kategorijama, a od toga najviše nad *"Slate"*, *"Face-to-face"* i *"Verbatim"*.

Zanimljivo je promotriti **performanse SNLI modela** nad pojedinom kategorijom u odnosu na združeni model. Kategorije nad kojima SNLI model postiže *ispodprosječne rezultate* upravo su kategorije nad kojima združeni model postiže *najveća poboljšanja*, kao što su *"Fiction"*, *"Slate"* (rast od >1%) te *"Verbatim"* (rast 1.6%). S druge strane, kategorije nad kojima je SNLI model *najsupješniji* (izuzev sâmog SNLI-a) pokazuju *jednake ili lošije* rezultate kod združenog modela, npr. *"Government"* (pad od 0.7%), *"Travel"* (pad od 1%) ili *"Letters"* (minimalno poboljšanje od 0.3%, obzirom da je to najuspješnije evaluirana kategorija SNLI treniranog modela s čak 63.02% točnosti).

U usporedbi s ESIM modelom iz [52], premda je polazni Keras ESIM nešto lošiji u performansama nad SNLI-em kao što je vidljivo u tablici 4.3, isti model treniran nad MNLI skupom već na njemu u pojedinim kategorijama postiže bolje rezultate nego ESIM iz [52], posebno *"Telephone"* (0.6%) te *"Government"* (2.2% bolji rezultat).

Upravo te dvije kategorije pate od pada performansi kod združenog modela, no poboljšanje nad *"Government"* dovoljno je da i s tim lošijim performansama model bude uspješniji nego [52], a združenom modelu rastu performanse nad *"Face-to-face"* preko uspješnosti modela iz [52] (1% bolji rezultat). Doduše, treba uzeti u obzir da su rezultati u tablici za model iz [52] treniran nad MNLI, a ne združenim skupom. Također, treba uzeti u obzir da su sve navedene performanse iz rada [52] performanse na testnom skupu, dok su ovdje izneseni rezultati, premda validirani tijekom treninga samo funkcijom troška, ipak testirani nad validacijskim skupom.

Unaprijeđenje rezultata

Postoji nekoliko pristupa kojima se mogu unaprijediti rezultati. Istaknuta razlika SNLI-a i MNLI-a je razlika u vokabularu (SNLI 56K riječi naspram MNLI 175K riječi). Pristup poboljšanju na tom području bio bi uključivanje čitavog vokabulara SNLI-a u matricu projekcija riječi koje se treniraju zajedno s parametrima modela, prilikom združenog treniranja MNLI i SNLI skupova. Na taj način, treniranjem s jednako konstruiranim trening skupom (MNLI + 15% SNLI-a) osiguralo bi jednaku zastupljenost rečenica iz svake od kategorija, a model bi dobio prednost proširenog vokabulara.

Određeni potencijal imaju i različite kategorije MNLI-a i prijenos znanja modela treniranog nad nekima od njih na druge. Budući da MNLI dolazi distribuiran s kategorijom kojoj pripada svaki par rečenica, moguće je trenirati model nad pojedinom kategorijom i isprobati prijenos znanja između sličnih (npr. telefonski razgovori i razgovori licem u lice) i različitih (razgovori licem u lice naspram vladinih terorističkih izvještaja). Ovo je koncept koji je djelomično istražen u samom radu [52]. Nad njime bi se također dalo graditi gore preporučenim poboljšanjima proširivanjem vokabulara bez povećanja trening skupa.

Naposljetku, za značajnije zaključivanje o rezultatima, potrebno je napraviti statističke testove koji uspoređuju značajnost razlika modela treniranih nad različitim skupovima.

Poglavlje 5

Zaključak

Polazna manjkava ESIM implementacija ([55]) usklađena je nizom prilagodbi kako bi odgovarala originalnom ESIM modelu u [11]. Korištena je Keras ESIM implementacija jer Keras predstavlja jednostavno, objektno orijentirano sučelje za rad s neuronskim mrežama.

Nad ispravljenim Keras ESIM modelom napravljena je Bayesova optimizacija hiperparametara gdje je postignuto poboljšanje performansi. Premda povećanje performansi nije preveliko, korisna posljedica Bayesove optimizacije jest uvid u utjecaj pojedinih hiperparametara na performanse modela. Izdvojeni su hiperparametri koji više utječu na uspjeh modela u odnosu na one koji utječu manje. To je rezultat na kojem se može graditi dalje, nastavljajući Bayesovu optimizaciju drugih hiperparametara te novih vrijednosti za postojeće.

Prijenos znanja pokazao je zanimljive rezultate: treniranje modela nad SNLI rezultira gotovo jednakom točnošću nad MNLI skupom kao isti model treniran nad MNLI i testiran nad SNLI. S druge strane, vidljivo je da dodavanje podskupa SNLI skupa MNLI-u, što uvodi dio jezične strukture SNLI-a uz postojeće kategorije, rezultira poboljšanjem nad 7/10 kategorija MNLI-a te SNLI-em. Posebno je istaknuto kako su dodavanjem SNLI-a poboljšane performanse nad svim neviđenim kategorijama, nad nekima po 1.5%, premda SNLI sadrži općenito drugačiji vokabular i jezičnu strukturu od MNLI-a.

Općenito, kombinacija MNLI i SNLI skupova čini se obećavajućom. Jezična i tematska raznolikost MNLI-a garantira njegovu upotrebljivost pri stvaranju budućih modela obrade prirodnog jezika. S druge strane, eksperimenti su pokazali loše performanse MNLI-a nad SNLI testnim skupom ukazujući na nedostatnost MNLI-a da samostalno modelira jezik korišten u SNLI skupu. Time je istaknut doprinos SNLI-a, kako za povećanje performansi nad SNLI testnim skupom, tako i nad drugim, MNLI specifičnim kategorijama. To znači da SNLI i dalje ima bitnu ulogu kao trening skup te predstavlja koristan dodatak MNLI-u, kao dodatna kategorija uz postojeće kategorije MNLI-a.

Sveukupno, iako počinjemo od nešto slabijeg modela (u usporedbi s drugim ESIM im-

plementacijama mjereno nad SNLI-em), na nekolicini metrika dostižemo i bolji rezultat nego [52]. Obzirom da je u tom radu korišten gotov model ESIM iz [11] bez (navedenih) promjena na hiperparametre, mogući uzrok boljih performansi su optimizirani hiperparametri Bayesovom optimizacijom ako su sve ostale komponente tih modela jednake (do na implementacijski okvir Keras odnosno TensorFlow).

Ovdje postignuti rezultati ilustriraju primjenu dubokih modela neuronskih mreža na problemu prepoznavanja slijednosti i predstavljaju potencijalni temelj za eventualne buduće eksperimente na tom području, s istaknutim prijedlozima za unaprijeđenje ovdje postignutih rezultata.

Bibliografija

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu i Xiaoqiang Zheng, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, 2015, `https://www.tensorflow.org/`, Software available from tensorflow.org.

[2] Sam Abrahams, Danijar Hafner, Erik Erwitt i Ariel Scarpinelli, *TensorFlow for Machine Intelligence: A Hands-on Introduction to Learning Algorithms*, Bleeding Edge Press, 2016.

[3] Dzmitry Bahdanau, Kyunghyun Cho i Yoshua Bengio, *Neural machine translation by jointly learning to align and translate*, arXiv preprint arXiv:1409.0473 (2014).

[4] Yoshua Bengio, *Learning deep architectures for AI*, Foundations and Trends in Machine Learning **2** (2009), br. 1, 1–127, Also published as a book. Now Publishers, 2009.

[5] ———, *Practical recommendations for gradient-based training of deep architectures*, Neural networks: Tricks of the trade, Springer, 2012, str. 437–478.

[6] James S Bergstra, Rémi Bardenet, Yoshua Bengio i Balázs Kégl, *Algorithms for hyper-parameter optimization*, Advances in neural information processing systems, 2011, str. 2546–2554.

[7] James Bergstra i Yoshua Bengio, *Random search for hyper-parameter optimization*, Journal of Machine Learning Research **13** (2012), br. Feb, 281–305.

[8] James Bergstra, Daniel Yamins i David Daniel Cox, *Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures*, (2013).

[9] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang i ost., *End to end learning for self-driving cars*, arXiv preprint arXiv:1604.07316 (2016).

[10] Samuel R Bowman, Gabor Angeli, Christopher Potts i Christopher D Manning, *A large annotated corpus for learning natural language inference*, arXiv preprint arXiv:1508.05326 (2015).

[11] Qian Chen, Xiaodan Zhu, Zhenhua Ling, Si Wei, Hui Jiang i Diana Inkpen, *Enhanced lstm for natural language inference*, arXiv preprint arXiv:1609.06038 (2016).

[12] Jianpeng Cheng, Li Dong i Mirella Lapata, *Long short-term memory-networks for machine reading*, arXiv preprint arXiv:1601.06733 (2016).

[13] Gennaro Chierchia i Sally McConnell-Ginet, *Meaning and grammar: An introduction to semantics*, MIT press, 2000.

[14] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk i Yoshua Bengio, *Learning phrase representations using RNN encoder-decoder for statistical machine translation*, arXiv preprint arXiv:1406.1078 (2014).

[15] François Chollet i ost., *Keras*, `https://keras.io`, 2015.

[16] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho i Yoshua Bengio, *Attention-based models for speech recognition*, Advances in neural information processing systems, 2015, str. 577–585.

[17] Daoud Clarke, *Creating a Textual Entailment Dataset Automatically from Guardian Articles*, 2011, `https://github.com/daoudclarke/rte-experiment`, posjećeno 28.10.2018.

[18] George Cybenko, *Approximations by superpositions of a sigmoidal function*, Mathematics of Control, Signals and Systems **2** (1989), 303–314.

[19] Ido Dagan, Bill Dolan, Bernardo Magnini i Dan Roth, *Recognizing textual entailment: Rational, evaluation and approaches–erratum*, Natural Language Engineering **16** (2010), br. 1, 105–105.

[20] Ido Dagan, Oren Glickman i Bernardo Magnini, *The PASCAL Recognising Textual Entailment Challenge*, Machine Learning Challenges (2006), 177.

[21] Andrew M. Dai, Christopher Olah i Quoc V. Le, *Document embedding with paragraph vectors*, NIPS Deep Learning Workshop, 2015.

[22] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le i ost., *Large scale distributed deep networks*, Advances in neural information processing systems, 2012, str. 1223–1231.

[23] Arden Dertat, *Applied Deep Learning - Part 1: Artificial Neural Networks*, 2017, `https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6`, posjećeno 20.10.2018.

[24] Timothy Dozat, *Incorporating nesterov momentum into adam*, (2016).

[25] John Duchi, Elad Hazan i Yoram Singer, *Adaptive subgradient methods for online learning and stochastic optimization*, Journal of Machine Learning Research **12** (2011), br. Jul, 2121–2159.

[26] Oren Glickman, Ido Dagan i Moshe Koppel, *A lexical alignment model for probabilistic textual entailment*, Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment, Springer, 2006, str. 287–298.

[27] Xavier Glorot, Antoine Bordes i Yoshua Bengio, *Deep sparse rectifier neural networks*, Proceedings of the fourteenth international conference on artificial intelligence and statistics, 2011, str. 315–323.

[28] Ian Goodfellow, Yoshua Bengio i Aaron Courville, *Deep Learning*, MIT Press, 2016, `http://www.deeplearningbook.org`.

[29] Sepp Hochreiter i Jürgen Schmidhuber, *Long short-term memory*, Neural computation **9** (1997), br. 8, 1735–1780.

[30] Kurt Hornik, Maxwell Stinchcombe i Halbert White, *Multilayer feedforward networks are universal approximators*, Neural networks **2** (1989), br. 5, 359–366.

[31] J. D. Hunter, *Matplotlib: A 2D graphics environment*, Computing In Science & Engineering **9** (2007), br. 3, 90–95.

[32] Frank Hutter, Holger H Hoos i Kevin Leyton-Brown, *Sequential model-based optimization for general algorithm configuration*, International Conference on Learning and Intelligent Optimization, Springer, 2011, str. 507–523.

[33] Kevin Jarrett, Koray Kavukcuoglu, Yann LeCun i ost., *What is the best multi-stage architecture for object recognition?*, Computer Vision, 2009 IEEE 12th International Conference on, IEEE, 2009, str. 2146–2153.

[34] Diederik P Kingma i Jimmy Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980 (2014).

[35] Yann LeCun, Yoshua Bengio i Geoffrey Hinton, *Deep learning*, nature **521** (2015), br. 7553, 436.

[36] Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, Roberto Zamparelli i ost., *A SICK cure for the evaluation of compositional distributional semantic models.*, LREC, 2014, str. 216–223.

[37] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado i Jeff Dean, *Distributed representations of words and phrases and their compositionality*, Advances in neural information processing systems, 2013, str. 3111–3119.

[38] Tom M Mitchell i ost., *Machine learning*, McGraw-Hill, 1997.

[39] Ankur P Parikh, Oscar Täckström, Dipanjan Das i Jakob Uszkoreit, *A decomposable attention model for natural language inference*, arXiv preprint arXiv:1606.01933 (2016).

[40] Jeffrey Pennington, Richard Socher i Christopher Manning, *Glove: Global vectors for word representation*, Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), 2014, str. 1532–1543.

[41] Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ i Phil Blunsom, *Reasoning about entailment with neural attention*, arXiv preprint arXiv:1509.06664 (2015).

[42] Sebastian Ruder, *An overview of gradient descent optimization algorithms*, arXiv preprint arXiv:1609.04747 (2016).

[43] David E Rumelhart, Geoffrey E Hinton i Ronald J Williams, *Learning representations by back-propagating errors*, nature **323** (1986), br. 6088, 533.

[44] Alexander M Rush, Sumit Chopra i Jason Weston, *A neural attention model for abstractive sentence summarization*, arXiv preprint arXiv:1509.00685 (2015).

[45] Helmut Schmid, *Probabilistic part-ofispeech tagging using decision trees*, New methods in language processing, 2013, str. 154.

[46] Sagar Sharma, *Activation Functions: Neural Networks*, 2017, `https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6`, posjećeno 18.10.2018.

[47] Jasper Snoek, Hugo Larochelle i Ryan P Adams, *Practical bayesian optimization of machine learning algorithms*, Advances in neural information processing systems, 2012, str. 2951–2959.

[48] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever i Ruslan Salakhutdinov, *Dropout: a simple way to prevent neural networks from overfitting*, The Journal of Machine Learning Research **15** (2014), br. 1, 1929–1958.

[49] Ilya Sutskever, James Martens, George Dahl i Geoffrey Hinton, *On the importance of initialization and momentum in deep learning*, International conference on machine learning, 2013, str. 1139–1147.

[50] Ilya Sutskever, Oriol Vinyals i Quoc V Le, *Sequence to sequence learning with neural networks*, Advances in neural information processing systems, 2014, str. 3104–3112.

[51] Theano Development Team, *Theano: A Python framework for fast computation of mathematical expressions*, arXiv e-prints **abs/1605.02688** (2016), `http://arxiv.org/abs/1605.02688`.

[52] Adina Williams, Nikita Nangia i Samuel Bowman, *A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference*, Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), Association for Computational Linguistics, 2018, str. 1112–1122, `http://aclweb.org/anthology/N18-1101`.

[53] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel i Yoshua Bengio, *Show, attend and tell: Neural image caption generation with visual attention*, International conference on machine learning, 2015, str. 2048–2057.

[54] Peter Young, Alice Lai, Micah Hodosh i Julia Hockenmaier, *From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions*, Transactions of the Association for Computational Linguistics **2** (2014), 67–78.

[55] Jingkuan Zhang, *SNLI-RTE in Keras*, 2017, https://github.com/adamzjk/SNLI-Keras, posjećeno 10.08.2018.

Sažetak

U ovom radu obrađen je problem prepoznavanja slijednosti teksta, kao bitne komponente mnogih drugih problema obrade prirodnog jezika. Opisan je pristup rješavanju problema tehnikama dubokog učenja. Istaknute su prednosti modela dubokog učenja u odnosu na tradicionalno oblikovane modele strojnog učenja s ručnim probirom značajki. Izneseni su osnovni koncepti strojnog učenja potrebni za razumijevanje izloženog. Prezentirana su tri znanstvena rada koji predstavljaju najnovije tehnike rješavanja zadatka slijednosti. Radovi su razloženi po cjelinama, analizirani i uspoređeni međusobno. Time je pružen pregled suvremenih pristupa rješavanju problema slijednosti dubokim učenjem. Posljednji dio predstavlja praktičan aspekt rada, s programskom implementacijom prezentiranog ESIM modela, u drugom programskom okviru (Keras). Kao polazna korištena je tuđa implementacija bazirana na ESIM modelu, koja inicijalno nije bila usklađena s arhitekturom originalnog ESIM-a. Dorađena Keras ESIM implementacija usklađena je s arhitekturom originalnog ESIM-a te postiže rezultate blizu rezultata originalnog modela.

Provedena je Bayesova optimizacija hiperparametara nad modelom, što je rezultiralo poboljšanjem performansi. Resultati optimizacije su predstavljeni i analizirani. Također, Bayesova optimziacija istaknula je doprinos pojedinih hiperparametara na uspjeh modela u odnosu na druge, što predstavlja dobru osnovu za nastavak rada nad ovim rezultatima. Model je testiran nad novo objavljenim skupom podataka velikog volumena, MNLI, te su postignuti rezultati blizu onima referentnog ESIM modela iz tog rada. Naposljetku, testiran je prijenos znanja modela na različite skupove i analizirani su rezultati po kategorijama. U usporedbi s referentnim modelom, ovdje korišten Keras ESIM postiže općenito tek nešto niže performanse, prestižući ga u nekim kategorijama. Također, pokazana su poboljšanja ESIM modela treniranog nad združenim MNLI i SNLI skupovima u odnosu na treniranje samo nad jednim skupom. Premda MNLI skup predstavlja bogat i raznolik skup za treniranje dubokih modela u odnosu na SNLI, pokazan je i značajan doprinos SNLI skupa performansama modela u odnosu na treniranje samo nad MNLI-em.

Summary

This thesis presents work on recognizing textual entailment (RTE), a relevant subtask for many other natural language processing tasks. Described herein is the approach using deep learning with emphasized benefits of using deep models in contrast to traditionally designed machine learning models using hand-crafted features. Basic concepts of machine learning are explained that are needed for the understanding of presented matter. Three scientific papers are presented which contain novel techniques for textual entailment recognition using deep learning models. The papers are explained. analysed and compared by parts, giving an overview of contemporary approaches to natural language inference using deep learning. The last part of the thesis represents the practical aspect of work, with an implementation of described ESIM model in a different framework (Keras). Existing implementation of said ESIM model was used as a starting point, which differed from the original code. After additional work of aligning that implementation with the original ESIM model code, resulting Keras ESIM achieved results close to the results of the original ESIM model and was used further on.

Performance of the model was improved using Bayes optimization of hyperparameters, results of which were presented and analyzed. Beside improving results, Bayes optimization highlighted some hyperparameters as more contributing than others to the model performance, laying ground for further work upon these results. Model was also tested on newly released dataset for natural language inference, MNLI, achieving results comparable to reference ESIM model from MNLI paper. Lastly, knowledge transfer was tested across multiple datasets with results analyzed by category. Compared to the reference model, this thesis' Keras ESIM achieves only marginally lower performance across the board, even surpassing the reference ESIM in some categories. Also, jointly trained ESIM on a combination of MNLI and SNLI datasets shows performance improvements compared to a model trained on a single dataset. Despite MNLI being richer and more language diverse dataset for training deep models compared to SNLI, a significant improvement of performance is demonstrated using a combination of datasets, showing a contribution of SNLI corpus for model training.

Životopis

Dinko Ždravac rođen je 17.09.1992. u Zagrebu. Nakon 3 završena razreda OŠ grofa Janka Draškovića u Zagrebu, osnovnoškolsko obrazovanje nastavio je u OŠ Pušća u Donjoj Pušći. Srednjoškolsko obrazovanje nastavio je programom opće gimnazije u gimnaziji Lucijana Vranjanina, nakon koje upisuje PMF u Zagrebu. 2016. završava preddiplomski studij podučavanja matematike te upisuje diplomski studij, smjer Računarstvo i matematika.

Još od osnovne škole pokazivao je interes za matematiku sudjelovanjem na natjecanjima 4 godine zaredom. Za vrijeme studija radio je u Hrvatskom Telekomu, Iskonu, A1 (nekadašnji Vipnet) i Infodomu, gdje radi u trenutku pisanja diplomskog rada, na bazama podataka i razvoju ERP rješenja za poslovne korisnike. Tijekom trajanja diplomskog studija pokazuje interes za područje umjetne inteligencije i strojnog učenja, rješavanjem problema predstavljenih na Kaggle natjecanjima u sklopu seminarskih radova iz kolegija Umjetna inteligencija (*"Predicting altruism through free pizza"*) i Strojno učenje (*"Daily News for Stock Market Prediction"*), interes koji je nastavio odabirom teme diplomskog rada iz područja dubokog učenja.